UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

HelpStack: Una herramienta de consulta integrada en el entorno de desarrollo Eclipse

Autor: Juan Conejo Laguna
Tutor: Juan De Lara Jaramillo

MARZO 2020

HelpStack: Una herramienta de consulta integrada en el entorno de desarrollo Eclipse.

AUTOR: Juan Conejo Laguna
TUTOR: Juan De Lara Jaramillo

Grupo de modelado e ingeniería del software (miso)
Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
marzo de 2020

Resumen (Castellano)

El lenguaje de programación Java es uno de los más utilizados para el desarrollo de software debido a sus características de multiplataforma y compatibilidad con la gran mayoría de dispositivos, así como la gran documentación y apoyo de la comunidad que existe en los diversos foros.

Para el desarrollo en este lenguaje, Eclipse es considerado uno de los entornos de desarrollo más apropiados y el que los estudiantes o los que comienzan a desarrollar con Java se instalan y empiezan a familiarizarse.

Aunque Eclipse aporta una gran cantidad de ayuda al programador, siempre existirá un hábito en cualquiera de ellos por el cual se minimiza la ventana del entorno, se lanza el navegador y se busca una duda relacionada con el lenguaje. Para finalizar el proceso, en muchas ocasiones se intenta buscar un resultado en foros especializados en resolver dicho tipo de dudas, y en concreto Stack Overflow.

Este Trabajo Fin de Grado consiste en el desarrollo de una herramienta para la consulta en foros tales como Stack Overflow o bases de datos internas integrada en el entorno de programación Eclipse. A lo largo del documento se analizarán diferentes tecnologías, patrones y diseños visuales, todo enfocado a la comodidad visual del usuario, así como un eficiente proceso de búsqueda de información.

La integración en el entorno de desarrollo juega un papel fundamental en este proyecto, ya que, se incluirán funcionalidades alternativas a la simple búsqueda de información. La inclusión de una base de datos interna para la creación de preguntas personalizadas ayudará a que los resultados sean más relevantes al proyecto en el que se esté trabajando. Por otro lado, la búsqueda de códigos proporcionará un vistazo general a todos los códigos relevantes relacionados con la consulta. La herramienta será evaluada de cara a su usabilidad y se obtendrán buenos resultados.

Así mismo se pretende mostrar cómo esta herramienta se podría expandir de manera que otros foros o servicios puedan incorporarse a la búsqueda y así poder tener resultados más relevantes.

Abstract (English)

The Java programming language is one of the most used for software development due to its multiplatform characteristics and compatibility with the vast majority of devices, as well as the great documentation and community support that exists in the various forums.

For the development in this language, Eclipse is considered one of the most appropriate development environments and the one that students or those who start developing with Java install and start to get familiar with.

Although Eclipse provides a great deal of help to the programmer, there will always be a habit in any of them whereby the environment window is minimised, the browser is launched and a query related to the language is sought. To finish the process, in many occasions we try to look for a result in specialized forums to solve this kind of doubts, and specifically Stack Overflow.

This Bachelor Thesis consists of the development of a tool for the query in forums such as Stack Overflow or internal databases integrated in the Eclipse programming environment. Throughout the document we will see different technologies, patterns and visual designs, all focused on the visual comfort of the user, as well as an efficient information search process.

The integration in the development environment plays a fundamental role in this project, since, alternative functionalities to the simple information search will be included. The inclusion of an internal database for the creation of customized questions will help make the results more relevant to the project being worked on. On the other hand, the code search will provide an overview of all relevant codes related to the query.

This document will also explain how this tool could be expanded so that other forums or services can be incorporated into the search and thus have more relevant results.

Palabras clave (Castellano)

Java, Eclipse, Plugin, Consulta, Stack Overflow, Búsqueda, MongoDB, bases de datos, Patrones de diseño

Keywords (English)

Java, Eclipse, Plugin, Query, Stack Overflow, Search, MongoDB, Databases, Pattern Design

Agradecimientos

A Juan de Lara Jaramillo, profesor en la Escuela Politécnica Superior de la Universidad Autónoma de Madrid por su apoyo y consejo durante el desarrollo de este trabajo.

A mi familia por su comprensión y paciencia a lo largo de estos años en la carrera.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE LISTADOS

1 Introducción

1.1 Motivación

Al comenzar o retomar un lenguaje de programación es costumbre buscar aquellas dudas que surgen en foros. También cuando surge un error específico o un fallo de compilación nos ayudamos entre la comunidad para resolver esos problemas. Por otra parte, todas estas consultas suponen una ruptura en el flujo de trabajo al usar programas de terceros ajenos al entorno de desarrollo como el navegador.

Muchas veces simplemente se necesita encontrar aquel código o recordar aquel algoritmo que uno no se acuerda y no presta atención a la explicación adjunta. Otras veces interesa realizar algún apunte o publicar el estilo estandarizado de programación que se utiliza para que los que trabajan en un equipo puedan consultarlo a voluntad.

Con todo esto se propone el desarrollo de una herramienta que incorpore todas estas funcionalidades integrándose en el propio entorno de desarrollo. De esta manera se consigue enfocar el objetivo de lo que se quiere programar apoyándose en consultas complementarias y no en una búsqueda exhaustiva de las dudas que surgen.

En concreto, el entorno de programación va a ser Eclipse ya que está diseñado de una manera modular y con facilidades para desarrollar plugins o módulos que aprovecharán sus funcionalidades nativas, así como las desarrolladas por este proyecto.

1.2 Objetivos

- Comprender la manera en la que se integran los plugins dentro del entorno de desarrollo Eclipse
- Mostrar cómo aprovechar un servicio público para la implementación de una herramienta a través de su API.
- Convertir el proceso de consulta en una tarea sencilla que requiera únicamente el uso del entorno de desarrollo.
- Brindar un servicio interno para la gestión de las consultas asociadas a un proyecto específico.
- Exponer la información necesaria para una lectura legible teniendo en cuenta las necesidades comunes de los desarrolladores.

1.3 Organización de la memoria

En la sección 2 se explicará las diferentes herramientas y tecnologías que se van a usar para el desarrollo del plugin. En este apartado se hará más énfasis en el funcionamiento interno de Eclipse con relación a su capacidad de extenderse mediante plugins. Seguidamente se explicará la API REST de Stack Overflow y una breve introducción a MongoDB. Por último se mencionarán algunas librerias Java utilizadas durante el desarrollo del plugin y trabajos relacionados con objetivos similares a este proyecto.

En el apartado de diseño, escrito en la sección 3, se reflejará la infraestructura vista desde el entorno de desarrollo, la arquitectura ideada para el plugin y varios patrones de diseño utilizados para diferentes funciones dentro del proyecto. En esta sección se pretende dar un vistazo general al funcionamiento de los diferentes elementos presentes así como su conexión con el objetivo a realizar.

Una vez vistos los apartados anteriores se procederá a ahondar en el desarrollo propio del plugin en la sección 4. Primeramente se explicará la implementación del servicio que se encarga de comunicarse con el foro Stack Overflow. Después se abordará la idea de Intra Stack Exchange, un foro interno personalizable. Ambos servicios se incluirán en un motor de búsqueda que realizará las consultas formuladas por el usuario, el cual se explica cómo tercer apartado de esta sección. En última instancia se irá recorriendo todo lo referente al frontend de la herramienta, en este caso, vistas propias y extendidas del entorno de desarrollo Eclipse. En cada una de estas vistas se explicará el porqué de la elección de su diseño gráfico y el comportamiento de cada uno de los elementos que lo conforman.

Por último, para probar la utilidad y usabilidad de la herramienta de consulta se analizarán unos formularios contestados por un grupo de personas con los que obtener resultados que se analizarán para una valoración final.

2 Tecnologías usadas y Trabajo Relacionado

En esta sección se va a explicar los diferentes programas o tecnologías que están directamente relacionadas con el proyecto. En concreto se mencionarán: el entorno de desarrollo Eclipse así como de qué manera se integra este trabajo en él, el servicio Stack Exchange para la consulta de información, MongoDB para la persistencia de datos, librerías Java para un correcto uso de representaciones estándar de datos y por último aquellos trabajos relacionados en los que se discutirán las diferencias, ventajas y desventajas con respecto a este proyecto.

2.1 Eclipse

La utilización de un buen entorno de desarrollo es clave para el proceso de creación de cualquier proyecto involucrado, haciendo que el programador se sienta cómodo y ágil en tiempo de trabajo. Los IDEs proporcionan esta comodidad, incluyendo herramientas y diálogos de ayuda, avisos y errores de sintaxis en el código y muchas más funcionalidades que ayudan a minimizar los errores humanos que se comenten.

Eclipse es un IDE para el desarrollo principal del lenguaje de programación Java. Proporciona muchas ayudas en el código y atajos en el teclado, así como el autocompletado de código y plantillas de clases entre otras muchas cosas. Eclipse será el entorno usado para desarrollar la herramienta, ya que está codificado en Java y permite el lanzamiento del plugin en otra instancia independiente de la del desarrollo. Además, este entorno permite extenderse mediante el uso de plugins e integraremos nuestro proyecto en Eclipse desarrollando uno de estos.

2.1.1 OSGi y Equinox

Eclipse soporta la extensión de su funcionalidad mediante plugins, los cuales se acoplan al sistema del entorno y son capaces de funcionar con el ciclo de vida del software. Todo este mecanismo es gracias a que Eclipse es una implementación de la especificación OSGi, llamada Equinox. OSGi provee de una serie de requisitos que un software debe cumplir para satisfacer ciertos objetivos. El objetivo principal de OSGi es la creación de un programa compuesta por pequeños módulos o bundles, los cuales se comunican entre sí a través de los servicios. Equinox implementa OSGi y además crea nuevas funcionalidades como los puntos de extensión como alternativo al uso de los servicios.

Un punto de extensión provee de funcionalidades propias para ser extendido y usarlas para crear nuevas funcionalidades. Por ejemplo, para poder añadir una ventana a Eclipse con un diseño creado por nosotros primero debemos extender el punto de extensión con id *org.eclipse.ui.views*. Eclipse provee de gran variedad de puntos de extensión para crear menús, crear preferencias, crear editores, etc. En el anexo C se explica de manera más detallada la especificación OSGi y el funcionamiento de los puntos de extensión.

2.1.2 Eclipse PDE

Las funcionalidades nativas de Eclipse aportan diálogos para la creación de proyectos Java de consola, pero existen complementos o plugins como veremos más adelante que nos facilitan la creación de otros proyectos. En el caso del desarrollo de esta herramienta, se va a crear un proyecto de tipo plugin y, aunque bien es cierto que se podría partir desde un proyecto Java normal, se ha apoyado en el plugin Eclipse Plugin Development Environment (PDE).

Este plugin provee de herramientas y utilidades para construir otros plugins tales como, una interfaz interactiva de un fichero xml o plantillas de ejemplo de plugins más sencillos. Este complemento suele venir ya instalado en distribuciones de Eclipse diseñadas específicamente para desarrollar este tipo de programas, pero también se puede incorporar a un Eclipse ya existente.

2.1.3 SWT y JFACE

Standard Widget Toolkit es un conjunto de herramientas o librerías implementadas como clases e interfaces que, combinándolas, podemos generar una GUI. Este set de herramientas tiene el punto fuerte de ser multiplataforma e invariante ante distintos Sistemas Operativos. El programador puede estar seguro de desarrollar una interfaz gráfica que se verá igual en las distintas plataformas que soporte Java.

El objetivo de SWT es brindar la creación y el manejo de componentes gráficos nativos de un sistema operativo programáticamente. Cada sistema operativo brinda al usuario de elementos gráficos tales como botones, campos de selección, ventanas etc. Un botón creado con SWT en Windows se verá con el estilo de Windows y así respectivamente con cada sistema operativo.

También, se proporciona la gestión de eventos lo que permite al programador saber cuándo un usuario ejerce cierta acción sobre un componente, como clicar un botón o seleccionar un texto. SWT también brinda la posibilidad de crear tus propias figuras y dibujos por lo que se puede conseguir un aspecto personalizado.

Todas estas funcionalidades dan mucha flexibilidad a nuestras creaciones, pero también aportan mucha complejidad en proyectos grandes. Por eso, junto a SWT, se desarrolló JFace.

JFace simplifica el uso de componentes gráficos, ya que reconoce los hábitos y usos más comunes que implican la repetición de código y diseños más estandarizados. JFace utiliza los componentes y proporciona un acceso más cómodo y organizado para un mejor uso de la librería. Un ejemplo de esto es el manejo de los eventos. En SWT cada evento se maneja de manera individual mientras que en JFace provee de un objeto con el que se pueden aunar todos los eventos.

En el plugin a desarrollar se hará uso de algunos de los adaptadores de JFace tales como el TableViewer, que se encarga de mostrar información en forma de tabla y proporcionar eventos que se pueden capturar, como clicar en una fila, y asociarlo a un método. Anexo D se explica detalladamente el uso de SWT y JFace en Eclipse.

2.2 Stack Exchange

Stack Exchange es un compendio de, hasta la fecha, 173 comunidades en linea en donde lo habitual es la publicación de preguntas y respuestas entre usuarios. Se creó en 2010 y su objetivo siempre ha sido compartir el conocimiento y aprender entre la comunidad. Esta red es reconocida y confiable por todos los desarrolladores que la consultan y también forma parte de ella.

De entre todas las comunidades que oferta, posiblemente la más destacada y visitada es https://stackoverflow.com/. Esta web es sin duda una de las referentes en cuanto a consultas de desarrollo software hasta la fecha.

2.2.1 Stack Overflow

Cuando un desarrollador software tiene una duda, lo consulta en esta web ya que tiene la certeza de que, si alguien ya la ha resuelto, estará ahí. Este servicio está dedicado específicamente a obtener respuestas de la siguiente manera: El usuario publica una duda que tiene. Los demás usuarios podrán buscarla, tanto por su título o descripción como por sus etiquetas asociadas, y responderla.

Los usuarios también pueden votar las respuestas y la pregunta para saber a cuanta gente ha ayudado, pero lo más relevante es buscar si alguna respuesta ha sido aceptada por el creador de la pregunta, lo que significará que le ha ayudado y es información útil.

Los usuarios van construyendo el contenido del foro y se les va recompensado por su trabajo a través de recompensas y reputación.

2.2.2 API REST

Para añadirle flexibilidad y acceso, Stack Exchange proporciona una API desde la que se puede usar todas las funcionalidades tales como la búsqueda de preguntas con un cierto criterio o el manejo de la cuenta de un usuario para poder postear preguntas o responderlas.

En concreto, el servicio provee de una API REST con unas limitaciones iniciales para el correcto funcionamiento en un entorno donde se puede utilizar gratuitamente.

En general, las peticiones al servidor están relacionadas con la IP del cliente. Si dicho cliente genera 30 peticiones en un segundo, las posteriores no se tendrán en cuenta. Aunque ese límite de tiempo pueda cambiar, se considera que un flujo de tal magnitud supone un uso abusivo del servicio.

Otras restricciones anunciadas, tiene que ver con el uso de una cadena identificativa con el nombre de *access_token.* Si no se proporciona un *access_token*, la cuota del cliente estará basada en su IP y compartida con otras aplicaciones que usen el servicio con la misma IP. Se puede registrar una aplicación y obtener así una *key* para obtener un aumento en la cuota diaria. Si por el contrario se proporciona un *access_token*, la cuota estará basada en el par cliente/aplicación.

Al consultar la API REST, la respuesta estará contenida en un envoltorio o *wrapper* con información general como la lista de elementos devueltos, un flag que dirá si hay más elementos que enviar, la cuota actual y el número de peticiones que puedes realizar.

```
{
  "items": [
    {
      "included_fields": [],
      "filter_type": "safe",
      "filter": "none"
    }
  ],
  "has_more": false,
  "quota_max": 10000,
  "quota_remaining": 9997
}
```


Figura 1. Wrapper de una consulta. Los ítems son el resultado de dicha consulta.

Para consultar en el servicio es necesario hacer una petición GET a través del protocolo http con la dirección del método documentado. Existen diversos métodos que se utilizar para mostrar preguntas, respuestas, tags, usuarios. El método que vamos a utilizar para nuestro proyecto es el de la búsqueda avanzada llamada *search/advanced.*

2.3 MongoDB

Algunas funcionalidades de este proyecto requieren la persistencia de datos. Para ello se utiliza la base de datos no relacional MongoDB. La decisión de utilizar una base de datos no relacional es debido a la flexibilidad en la adición de campos sin tener que modificar una tabla como pasa en una base de datos relacional y el poder hacer uso de técnicas de búsqueda en varios campos tales con expresiones regulares.

En MongoDB existen diferentes estructuras parecidas a SQL para la organización del contenido. En primer lugar, se crea una base de datos, en la cual estará contenido las diferentes colecciones. Las colecciones son las equivalentes a las tablas de una base de datos relacional. Dentro de las colecciones residen los documentos, equivalentes también a los registros de una tabla en SQL.

Una característica importante mencionada anteriormente es la posibilidad de insertar documentos con diferentes campos, sin estar atados a un esquema como es habitual. Esto permite una gran flexibilidad al introducir nuevos documentos con nuevos campos y no estar modificando la colección entera. Cuando un documento se crea, MongoDB genera una cadena de identificación por defecto en el campo *_id.*

Por último, en este proyecto se hará uso de los *filtros* que proporciona esta base de datos para realizar las búsquedas de los documentos deseados.

2.4 Librerías Java

En este proyecto se va a hacer uso de varios formatos estándares tales como JSON, archivos de configuración INI y HTML. Para la correcta validación y manejo de estas estructuras, la comunidad de desarrolladores y empresas de renombre han desarrollado librerías que facilitan las tareas más comunes con estos archivos.

2.4.1 GSON

Trabajando con API REST lo más común es que los datos devueltos estén codificados en formato JSON y es aquí donde entra esta librería. GSON realiza una serialización directa de una cadena JSON a un objeto java. Esto permite la comodidad de evitar parsear la respuesta y los tipos en ella.

Su uso básico consiste en la llamada a los métodos *fromJson()* y *toJson()* en los que se pasarán por argumento la cadena o el objeto respectivamente y el tipo que se desea serializar. En la figura 10 se muestra un ejemplo del código del proyecto.

```
Type resultRestType = new TypeToken<ResultRest<Filter>>() {
}.getType();
ResultRest<Filter> result = gson.fromJson(response, resultRestType);
```

Figura 2. Conversión de json a objeto mediante el uso de gson.

Entre otras características GSON es capaz de excluir atributos de la clase para su conversión a JSON mediante la declaración del modificador *transient.* También, con la ayuda del decorador *@SerializedName,* se puede asociar un campo de JSON con diferente nombre al de la clase destinataria.

GSON fue desarrollada por Google, es código abierto y por tanto se puede hacer un uso libre de este software.

2.4.2 JSOUP

Al mostrar la información en Eclipse de las consultas se ha decidido usar el estándar HTML ya que facilita el estilo y la visualización del contenido. Para manejar este tipo de formato, JSOUP facilita mucho el trabajo. Además, los campos de la pregunta en la API REST se recibe en HTML con algunas etiquetas para diferenciar ciertos elementos importantes.

El uso que se le va a dar a esta librería comienza con parsear esos campos, lo que devolverá un *Document* o documento, el cual es una clase propia que implementa JSOUP. Con el objeto devuelto podemos empezar a buscar por clases, ids, atributos como si fuera una consulta de javascript cualquiera. Finalmente, obteniendo un iterador se podrá recorrer el resultado y modificar uno a uno cada elemento añadiendo clases o atributos a nuestro gusto. En la figura 11 se muestra un ejemplo de uso sacado de este proyecto.

```
public static String prettify(String content) {
    // Add prettify to code and pre tags
    Document doc = Jsoup.parse(content);
    Iterator<Element> iterator = doc.select("code, pre").iterator();

    while(iterator.hasNext()) {
        iterator.next().addClass("prettyprint");
    }

    return doc.html();
}
```

Figura 3. Añadiendo la clase prettyprint a los elementos con clases code y pre.

2.5 Proyectos Similares

Existen proyectos con un propósito parecido al descrito en este documento. A continuación, explicaremos algunos de estos programas y las principales diferencias con nuestra herramienta.

2.5.1 Seahawk

Integrando en Eclipse muchas facilidades que proporciona Stack Overflow, Seahaw [4] permite realizar consultas, mostrando los resultados en una lista y sus contenidos sobre una plantilla en una vista separada. En la figura 12 se muestra la interfaz de usuario de esta herramienta.

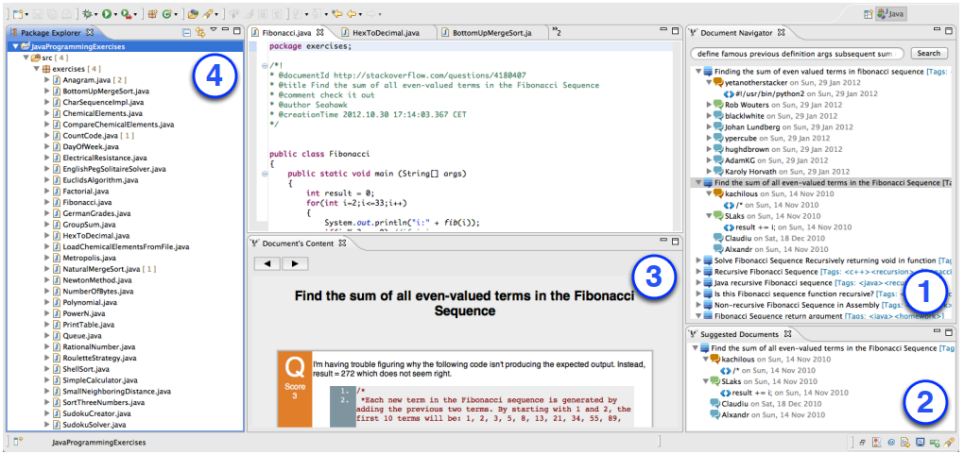

Figura 4. Interfaz de usuario de Seahawk [4]

Para la consulta del contenido en el foro hace uso de un servicio externo el cual posee un volcado de información de Stack Overflow. Para conseguir estos documentos desarrollaron un mecanismo de recolección de datos. Este mecanismo guarda los archivos en una base de datos relacional, los convierte en JSON y provee de una API REST bajo un servidor Apache. Este servicio se encargará de realizar las búsquedas y devolver los resultados todo ello a través del protocolo HTTP. Por tanto, la herramienta solo tendrá que realizar peticiones a este servicio y presentar los resultados en Eclipse.

La principal diferencia que se presenta con Seahawk es la manera en la que obtiene la información de Stack Overflow. En vez de crear un servicio que depende de otro servicio, en este proyecto hace uso de la propia API REST que proporciona el foro. Las búsquedas las implementa el propio servicio de Stack Overflow y no el plugin. De esta manera se genera un sistema más robusto y compacto frente a cambios en la infraestructura. También, HelpStack provee de otras funcionalidades como la búsqueda de códigos y una base de datos interna para la creación de preguntas personalizadas.

2.5.2 NLP2Code

NLP2code [5] propone que el proceso de consulta se realice en el propio editor de texto en vez de en una vista separada del entorno. Formulando la pregunta como si de código se tratara, el plugin mostrará un menú contextual que contendrá múltiples sugerencias para acabar la consulta planteada. Al seleccionar la opción deseada se insertará el código asociado. Si este código no fuera el deseado, mediante un botón o combinación de teclas se podrá cambiar de opción hasta que el resultado fuera satisfactorio.

Figura 5. Ciclo de funcionamiento de NLP2Code [5]

A diferencia del plugin a desarrollar, NLP2Code está enfocado en exclusivo a la búsqueda de códigos y omite toda la información asociada con el origen. También, en este trabajo de fin de grado se ha apostado por la organización en vistas y acceso a ellas mediante accesos rápidos con el teclado. Todo esto dota de mayor solución a un problema de carácter general no solo refiriéndose al código.

3 Diseño

Antes de explicar la implementación de la herramienta como tal primeramente debemos entender la arquitectura que sigue el programa y sus diferentes elementos que la componen. Además, se mostrará los diferentes patrones de diseño utilizados para estos elementos, en concreto, la creación de servicios, la técnica Modelo Vista Controlador y el uso de Wrappers para envolver características adicionales a las clases. Por último, se mostrará el diseño interno de la herramienta para observar la comunicación de sus diferentes módulos.

3.1 Arquitectura

Como se expuso en apartados anteriores, el plugin realiza diversas comunicaciones con servicios externos para recopilar información, ordenarla y posteriormente mostrarla de manera integrada en el entorno de desarrollo eclipse. La figura 14 muestra el esquema con estos servicios.

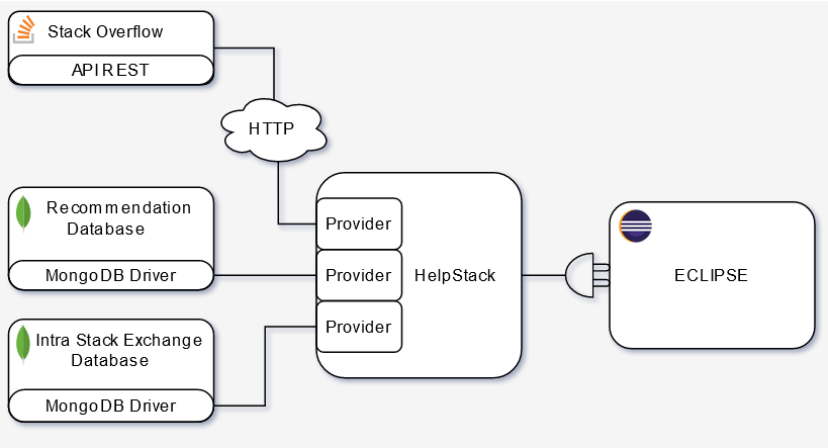

Figura 6. Arquitectura de la solución, que muestra la relación entre los servicios, Eclipse y HelpStack.

Stack overflow proporciona acceso a toda la información relacionada con el foro. En concreto se consultará las preguntas y respuestas deseadas. La comunicación se realizará mediante la API REST a través de internet con el protocolo HTTP.

La base de datos de recomendaciones almacenará aquellas preguntas que le han sido útil al usuario y que priorizará a la hora de mostrarlas en la vista integrada. De esta manera, cuando en un futuro el usuario busque la misma consulta u otra parecida se mostrará en las primeras posiciones de la tabla de resultados.

Intra Stack Exchange, en cambio, permite la creación de un foro interno. En este lugar se ubicarán las preguntas y respuestas creadas por los usuarios de HelStack. De esta manera se consigue que al buscar una duda específica del proyecto que se está desarrollando se obtenga a su vez una respuesta concreta y específica referenciada a ese proyecto. También serán prioritarias a la hora de presentar la información. Las dos bases de datos son no relacionales bajo MongoDB.

El plugin implementa una interfaz o proveedor por cada uno de estos servicios, el cual hará uso de sus API y proveerá de un acceso global a cada uno de estos dentro del código. Por tanto, estas interfaces se convertirán en servicios internos dentro del plugin los cuales explicaremos en el siguiente apartado.

Por último, se extenderá la funcionalidad de algunos puntos de extensión de Eclipse para el manejo de la información y su posterior visualización.

3.2 Diagrama de Clases

El diagrama de clases de esta herramienta se divide en varias secciones, y en concreto, en Servicios, Modelos, Vistas, Controladores y Manejadores de comandos.

Las clases encargadas de manejar los servicios externos a HelpStack se muestran en la figura 15. IntraSEProvider implementa métodos sobre la manipulación de la base de datos MongoDB dedicada a Intra Stack Exchange. StackOverflowProvider proveerá la capacidad de buscar en Stack Overlow a través de la API REST que provee el servicio. Por último, RecommendationProvider será la encargada de almacenar aquellas preguntas que quieran ser recordadas y priorizadas. Filter representa el modelo del filtro devuelto por la API REST de Stack Overflow y ResultRest<T> refleja la respuesta con el contenido envuelto de cualquier petición a Stasck Overflow en el que se ahondará en posteriores apartados.

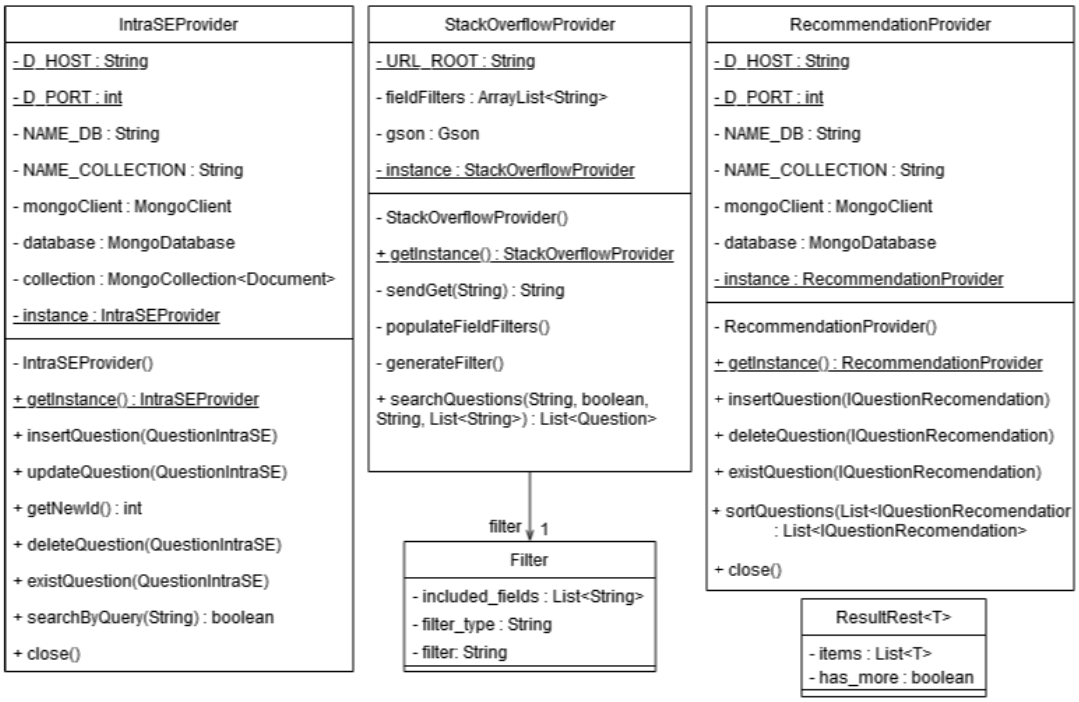

Figura 7. Diagrama de clases de los proveedores de servicios y sus dependencias

El modelo se compone de dos tipos de preguntas: Question, asociado al modelo de pregunta que devuelve el servicio Stack Overflow y QuestionIntraSE, referido al modelo de pregunta de Intra Stack Exchange.

Ya que usamos la librería Gson para la traducción de la respuesta de SO a objetos de java, se necesita definir las clases de la que depende la respuesta y sus campos. Para ello se define Answer y Question.

Ambas preguntas se asocian con su adaptador usado para relacionar el modelo con información adicional que ayudará a la unificación de estas clases en el proceso de búsqueda gracias a que implementan las tres interfaces ICodeSearch, IQuestionRecommendation, IQuestionSearch tal como muestra la figura 16.

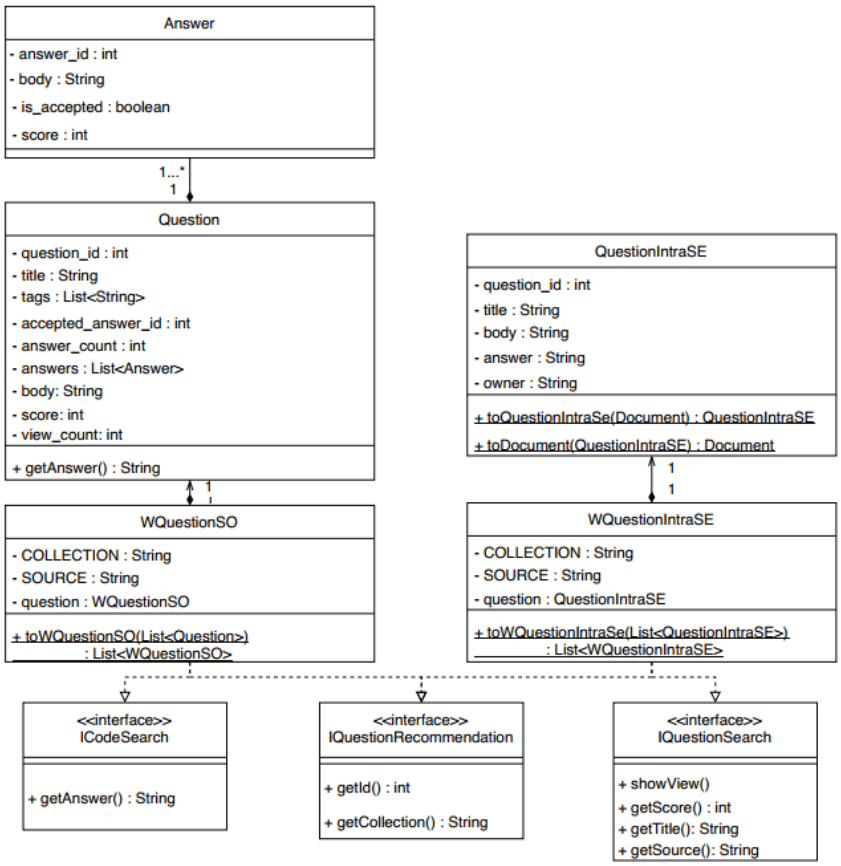

Figura 8. Diagrama de clases del modelo con sus adaptadores e interfaces implementadas.

Por otro lado, las vistas deben extender a una clase abstracta llamada ViewPart y los dialogos de Dialog para que puedan incorporarse a las vistas de eclipse. Las vistas que aparecen en la figura 17 son las correspondientes a la vista de búsqueda, búsqueda de códigos, creación de una nueva pregunta y los resultados de los servicios Stack Overflow e Intra Stack Exchange.

Si la vista tiene elementos gráficos a los cuales se les tienen que incorporar un comportamiento, tendrá asociado un controlador en cuyo método registerListeners() se ubicará todo el código dedicado a los escuchadores y manejadores de los elementos. En este método es donde se hará uso del motor de búsqueda SearchEngine el cual posee métodos estáticos para ser llamado desde cualquier parte del código si fuera necesario.

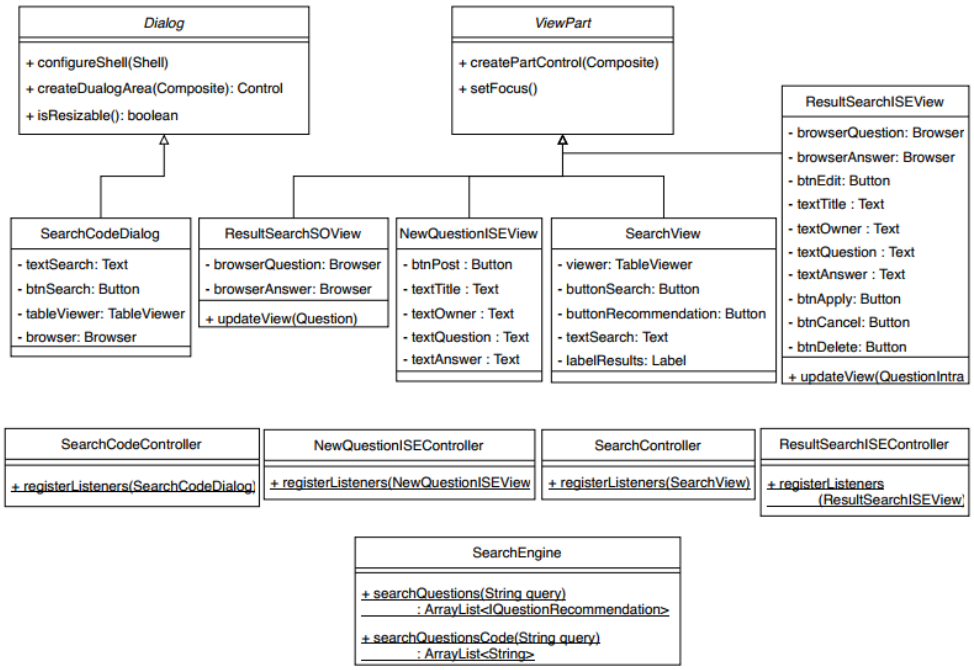

Figura 9. Diagrama de clases de las vistas y controladores de HelpStack.

Finalmente, para poder crear los comandos que se asociarán con menus a través de extensiones de puntos de extensión y que puedan abrir vistas y mostrarselas al usuario habrá que crear clases que extiendan a AbstractHandler como se muestra en la fiugura 18. Cada clase implementada se encarga de mostrar la vista de búsqueda, búsqueda de códigos y la creación de preguntas en Intra Stack Exchange.

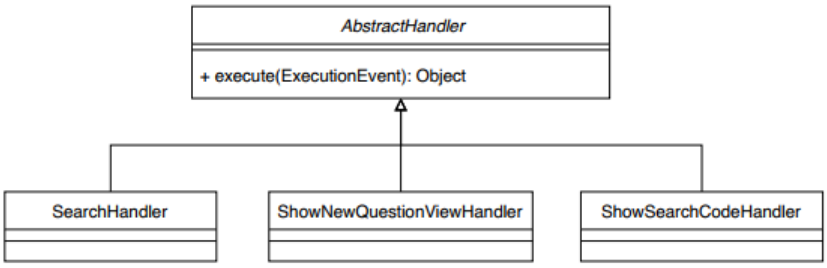

Figura 10. Diagrama de clases de las vistas y controladores de HelpStack.

3.3 Servicios

Como se mencionó anteriormente, el plugin implementa una interfaz para cada servicio externo. Esta interfaz o proveedor es la encargada de comunicarse con su servicio y realizar las operaciones deseadas por el usuario. Para un uso más cómodo de estas interfaces se han declarado globales mediante el patrón de diseño *Singleton*. El objetivo de esta decisión es convertir a estas interfaces en proveedores o servicios que en cualquier momento sean capaces de dar la información deseada.

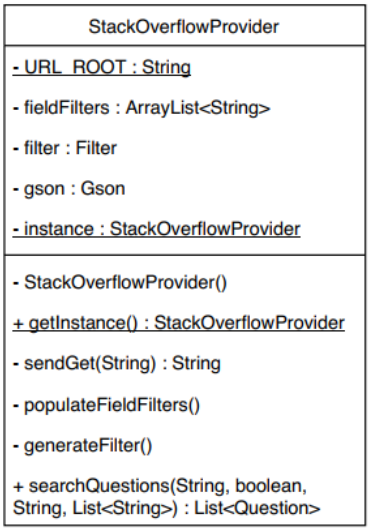

Figura 11. Proveedor de servicio Stack Overflow bajo el patrón Singleton

Para implementar un patrón *Singleton* se necesita crear una clase con un atributo del mismo tipo que el de la clase. Ya que no se debería poder acceder a este campo se cambiará su modificador de acceso a privado.

Desde fuera, si se pretende llamar a esta clase la primera invocación debe de ser el obtener la instancia. Por ello se crea el método *getInstance()* cuyo objetivo es siempre devolver la misma instancia y no estar creando varios objetos por cada llamada. La comprobación del atributo donde estará la única estancia decidirá si se debe crear el objeto o ya está creado.

El constructor también debe ser privado para controlar que solo exista una instancia de la clase, por lo que este constructor solo se llamara en la función *getInstance()*.

3.4 El Patrón Modelo Vista Controlador

El uso de las vistas en el entorno de programación Eclipse conlleva la implementación de una clase con una interfaz determinada. Esta clase es la encargada de crear todos los componentes gráficos y sus manejadores para interactuar con lo que quiere mostrar. Ya que el motor gráfico de Eclipse es SWT, la creación e inicialización de todos estos elementos supone la implementación de muchas líneas de código. Por tanto, se ha decidido usar el patrón de diseño Modelo Vista Controlador [6].

En este patrón tanto el modelo, la vista y el controlador están separados en distintas clases. El modelo hace referencia a aquella clase que contiene la información a mostrar en la vista. La vista contiene la creación de los elementos gráficos, así como su ubicación. Por último, el controlador será el único que pueda acceder tanto al modelo como a la vista para poder actualizar la vista según el modelo. En el caso de este proyecto, el controlador se encargará de registrar aquellas acciones asociadas a los botones y a la interacción del usuario con los elementos gráficos de forma general.

3.5 El Patrón Adaptador o Wrapper

En ocasiones nos interesa que el modelo lleve información asociada relacionada con la gestión que se realiza internamente, pero no queremos llenar de atributos y métodos que no tienen que ver dentro de la clase modelo. En este proyecto ocurre algo parecido, por una parte, se tiene la clase de la pregunta la cual alberga campos como título, cuerpo, autor, etc.

Y, por otra parte, se quiere implementar una interfaz y varios atributos que se usan para manejarlo en el módulo del motor de búsqueda.

Existe un patrón de diseño que ayuda a manejar este tipo de situaciones llamado Adaptador [7] o Wrapper. Un adaptador actúa como un envoltorio, conteniendo al modelo original, pero añadiendo más campos y métodos. La instancia que contiene la referencia mediante un atributo. En el constructor del envoltorio se pasará como argumento el modelo contenido. Para llamar a los métodos del modelo se crearán métodos personalizados en el envoltorio que llamarán al modelo, o bien, se podría implementar un método en el envoltorio que devolviera el modelo y ya con el modelo ir llamando a sus métodos. Un ejemplo del Wrapper se puede observar en la figura 20.

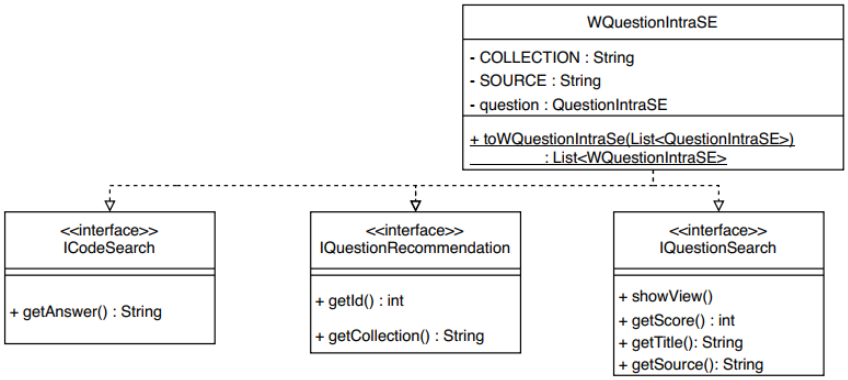

Figura 12. Ejemplo de patrón de diseño adaptador o Wrapper.

Un buen hábito de documentación es añadir algún tipo de identificación al envoltorio referenciando también al modelo que contiene. En este proyecto a los Wrappers se les suele nombrar empezando con la letra W y siguiendo con el nombre del modelo.

3.6 Diseño Interno

Hemos visto la infraestructura utilizada para comunicarnos con los servicios externos al plugin y a Eclipse. A continuación, veremos el diseño interno del plugin mostrada en la figura 21 y la inclusión de los diferentes diseños que vimos en apartados anteriores.

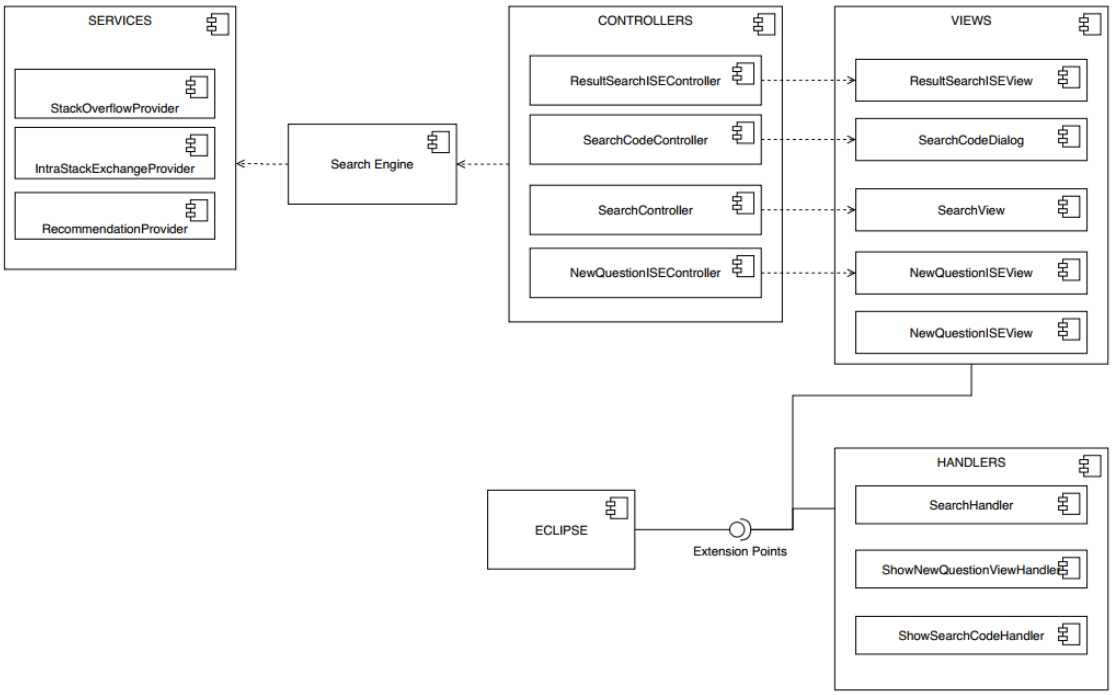

Figura 13. Diagrama de componentes de HelpStack.

En la zona de los **servicios** se ubican los proveedores globales implementados que aportan las diferentes funcionalidades del dominio al que están orientadas. En cada servicio o proveedor se definen los modelos que contienen la información en forma de objetos y que dichos servicios devolverán en sus métodos.

El **motor de búsqueda** se encargará de realizar las consultas en los distintos servicios, envolver las respuestas mediante *wrappers* que implementan interfaces comunes con las que puede agrupar todos resultados y devolver una lista única de preguntas provenientes de diferentes servicios.

En el grupo de las **vistas**, cada clase es la encargada de crear sus elementos gráficos contenidos y de actualizar su información dado su modelo. Por ejemplo, para mostrar la información de un resultado proveniente del servicio Stack Overflow, la clase *ResultSearchSOView* creará el layout SWT y añadirá un área de texto para el cuerpo de la pregunta y otro para el cuerpo de la respuesta. También implementará un método que poblará de información esos elementos dado su modelo.

En el caso de que la vista posea algún elemento gráfico con comportamiento reactivo a eventos, como por ejemplo un botón al seleccionarlo con el ratón, un **controlador** se encargará de implementar dichos comportamientos y los asociará a estos elementos gráficos al momento de la creación de la vista.

Para que las vistas estén contenidas en las ventanas nativas de Eclipse deben extender de los **puntos de extensión** correspondientes. Algunos de estos puntos de extensión también hacen referencia a los menús y botones de la interfaz de eclipse que ejecutan la funcionalidad implementada en los **manejadores o handlers**.

4 Desarrollo

El objetivo de este capítulo va a ser explicar más en detalle cada una de las partes que hemos mencionado en el apartado de Diseño y cómo se comunican entre sí. Para ello se va a dividir esta sección en 5 partes generales. En la primera parte se hablará de Stack Overflow, la API que proporciona este servicio y su integración con HelpStack. Como segunda parte se expondrá la idea de Intra Stack Exchange, un foro de consultas interno, y su implementación con el entorno de desarrollo. El motor de búsqueda estará en tercer lugar y se verá cómo se ha decidido unir los modelos de ambos servicios y el orden de prioridad de cada uno, así como la intervención del sistema de recomendados en esta fase. También se mostrarán los diseños de las vistas con sus respectivos elementos gráficos, su interacción con la parte lógica de HelpStack y los puntos de extensión usados de Eclipse para lograr su integración con este entorno de desarrollo. Por último, se explicará cómo realizar el despliegue y la exportación del plugin una vez finalizado el desarrollo.

4.1 API Stack Overflow

Como ya sabemos, Stack Overflow pone al servicio del usuario una API REST con el que consultar y manejar la información de este foro. Una API de este tipo se comunica bajo el protocolo HTTP.

Existen diversas librerías escritas en Java que ayudan y facilitan el manejo de este servicio (Cloud Rest Api Client [8], Stackoverflow java sdk [9], Jaxrs [10]). En el caso de Stackoverflow java sdk implementa todos los métodos de la API, esta librería permite consultas, búsquedas, identificación de usuario, publicar tus propias preguntas, comentar a cierto usuario etc. Ya que el alcance de este plugin solamente llega a la búsqueda de preguntas se ha decido implementar una propia librería a modo de servicio o proveedor para acceder a ella en cualquier ubicación del código.

Primero hay que saber cómo comunicarse con esta API REST. A continuación, tenemos un ejemplo de una petición a este servicio:

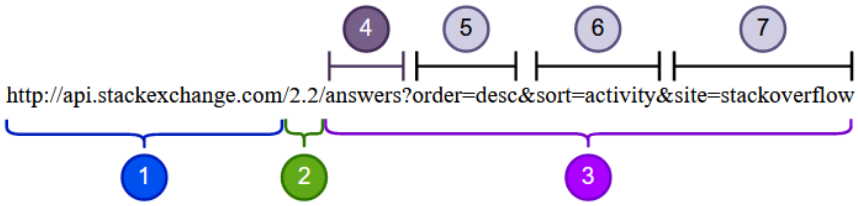

Figura 14. Ejemplo de llamada al método *answers* de la API de Stack Overflow.

Como podemos observar en la figura 22, se está realizando una petición GET con varios argumentos. Esta URL comienza con el nombre de dominio (1) asociado a los servidores de Stack Exchange reservando el subdominio *api* para referirse a este servicio en concreto. Seguidamente está marcada la versión de la API, en este caso 2.2 (2). Después de estas rutas se procede a anotar el método a usar con sus correspondientes parámetros (3). El usuario quiere mostrar todas las respuestas posibles, para ello hace uso del método *answers* (4) que tiene como parámetros el orden y el criterio de ordenación (5 y 6) y el sitio de donde sacar la información (7).

Por lo tanto, el resultado de esta petición serán todas las respuestas ubicadas en el dominio Stack Overflow, ordenadas de mayor a menor actividad.

4.1.1 Métodos usados

El principal objetivo del plugin es ser capaz de buscar preguntas basadas en una consulta, extraer el cuerpo de la pregunta y obtener el cuerpo de la respuesta aceptada. Para lograrlo se pueden plantear diversas soluciones:

1. Usar el método /*questions* el cual retornaría todas las preguntas de Stack Overflow y realizar la búsqueda de manera local en el plugin.
2. Usar el método /search el cual realiza la búsqueda en los servidores de Stack Overflow y devuelve las preguntas correspondientes.

La primera opción es tosca ya que hay un número alto de preguntas en el foro y sería inviable guardarlas en memoria para realizar búsquedas. Además, habría que implementar un motor de búsqueda con solo la información de las preguntas.

La segunda opción proporciona un algoritmo de búsqueda implementada en el lado del servidor por lo que no tendremos que crear uno propio. Aunque es la mejor alternativa sólo permite buscar por título por lo que es poco preciso.

El método */search/advanced* permite más parámetros para concretar más específicamente los criterios de la consulta. Estos parámetros son:

- q – Texto libre que coincidirá con todas las propiedades de la pregunta basados en un algoritmo.
- accepted – Booleano indicando, cuando se establece a *true*, que las preguntas devueltas tienen una respuesta aceptada por el creador de la pregunta.
- answers – Mínimo número de respuestas que la pregunta devuelta debe tener.
- body – Texto libre que debe aparecer en el cuerpo de la pregunta.
- closed – Cuando este parámetro contiene *true* solo buscará aquellas preguntas cerradas por el creador. En caso contrario buscará preguntas abiertas. Si se omite no se considera este criterio.
- migrated – Indica si la pregunta ha sido migrada a otro dominio.
- notice – Indica si debe devolver preguntas con avisos o no.
- nottagged – Lista de etiquetas separadas por el carácter ‘;’ que no deben aparecer en las etiquetas de las preguntas retornadas.
- tagged – Lista de etiquetas separadas por el carácter ‘;’ que deben aparecer en las etiquetas de las preguntas retornadas.
- title – Texto libre que debe aparecer en el título de las preguntas devueltas.
- user – Especifica el id del usuario que es el propietario de las preguntas.
- url – Url que aparece en la pregunta.
- views – Número mínimo de vistas que deben tener las preguntas.
- wiki – A *true* para devolver solo las preguntas pertenecientes a la wiki de la comunidad.
- sort – Criterio de ordenación de las preguntas. Los criterios a escoger son:
    - last_activity_date – Actividad
    - creation_date – Fecha de creación
    - score – Votos
    - relevance – Relevancia

Tras varias pruebas realizadas los mejores resultados se alcanzaban usando los parámetros *q, accepted, title, tagged* y *sort*. El criterio de ordenación para el parámetro *sort* es el de relevancia.

El resultado es una lista de preguntas representadas en JSON. Los campos por defecto que se muestran en una pregunta se pueden observar en la figura 23.


```
{
  "tags": [
    "java",
    "file-io",
    "ascii"
  ],
  "owner": {
    "reputation": 4721,
    "user_id": 130888,
    "user_type": "registered",
    "accept_rate": 100,
    "profile_image": "https://www.gravatar.com/avatar/dc215853f29bd2688
    "display_name": "Tim the Enchanter",
    "link": "https://stackoverflow.com/users/130888/tim-the-enchanter"
  },
  "is_answered": true,
  "view_count": 2283664,
  "protected_date": 1480304167,
  "accepted_answer_id": 4716521,
  "answer_count": 26,
  "score": 899,
  "last_activity_date": 1568629463,
  "creation_date": 1295288953,
  "last_edit_date": 1452863577,
  "question_id": 4716503,
  "link": "https://stackoverflow.com/questions/4716503/reading-a-plain-
  "title": "Reading a plain text file in Java"
}
```


Figura 15. Campos mostrados por defecto de una pregunta.

Muchos de los campos no nos interesan ya que no tienen utilidad y tampoco se van a mostrar al usuario. También se observa que faltan campos importantes como el cuerpo de la pregunta sin la cual el plugin carece de sentido.

Para decidir qué campos se quieren obtener y cuáles no, la API pone a disposición del desarrollador los *filtros*. Un filtro es una cadena de caracteres que codifica los campos a mostrar de diversos elementos y se pasa como argumento en una petición como cualquier otro parámetro. Si este parámetro es ignorado se aplica un filtro por defecto.

Por suerte, existe un método que nos permite crear nuestro propio filtro consiguiendo así los campos específicos que necesitamos. Este método se llama */filters/create*. Este método acepta los siguientes parámetros:

- include – Lista de campos que deben aparecer.
- exclude – Lista de campos que no deben aparecer.
- base – Filtro del que se parte para añadir mas campos o quitar algunos.

Ya que el filtro por defecto trae muchos campos no deseados se ha decidido partir de un filtro vacío llamado *none* y usar el parámetro *include* para incluir nuestros propios campos. Los campos seleccionados para el objeto **pregunta** son: id, título, etiquetas, id de la respuesta aceptada, número de respuestas, respuestas, el cuerpo, fecha de creación, booleano de si está respondida, votos y número de visitas. Para el objeto **respuesta**: id, cuerpo, booleano que indica si está aceptada y puntuación. El **envoltorio** que recubre todos los resultados tiene los campos: elementos y un booleano que indica si hay más resultados que no se muestran.

Para programar y crear todas estas funcionalidades se ha creado una clase llamada StackOverFlowProvider cuyos atributos son la url base *http://api.stackexchange.com/2.2/* marcada como una constante, los campos del filtro a crear y el objeto filtro.

Los métodos implementados son: la búsqueda de preguntas, el encargado de introducir los campos del filtro, el generador del filtro y por último el que realiza la petición HTTP mediante la librería nativa de Java *HttpURLConnection*.

Como se explicó anteriormente, esta clase está diseñada para ser accesible en cualquier lugar del código y por tanto está implementada bajo el patrón de diseño Singleton. En su constructor privado se crea el filtro deseado para poder usarse en futuras peticiones. Para crear el filtro primero hay que indicar que campos queremos personalizar de los objetos obtenidos de una petición, de esto se encarga el método populateFieldFilters(). Seguidamente se podrá llamar al método generateFilter() para crear el objeto de tipo Filter. Para buscar las preguntas se podrá realizar una llamada a searchQuestions() con los parámetros texto genérico, booleano que representa si la pregunta tiene una respuesta aceptada, un texto para el título y las etiquetas asociadas. Este método construirá la url asociada a la consulta y hará uso de sendGet() para realizar las peticiones HTTP requeridas.

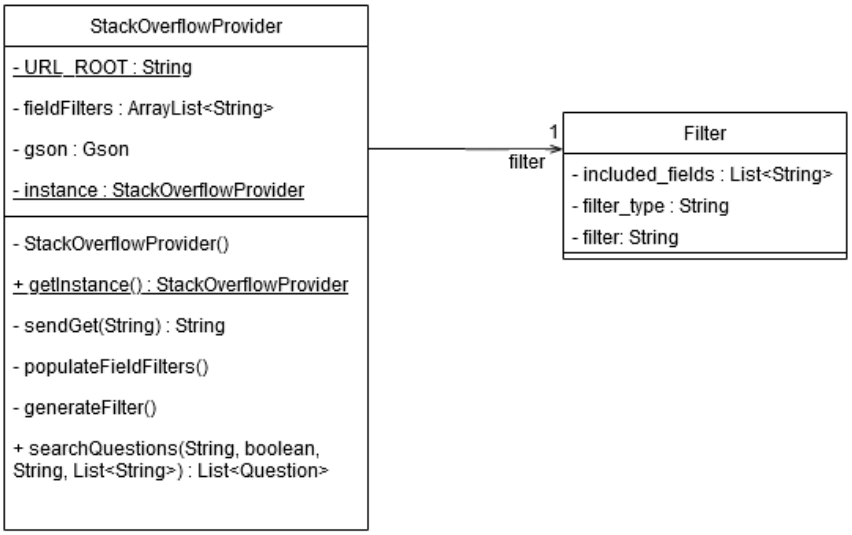

Figura 16. Clase StackOverflowProvider con sus atributos y métodos

4.1.2 GSON y modelos

Una vez hecha la parte funcional con la que nos podemos comunicar con el servicio de Stack Overflow, el siguiente paso es estructurar toda esa información a objetos manejables en Java.

En apartados anteriores se ha mencionado el uso de una librería que se encarga de transformar cadenas JSON a objetos Java llamada GSON. Al utilizar esta librería se debe diseñar una clase Java con los mismos atributos que el JSON de origen.

Ya que la respuesta de la API está envuelta con información adicional se ha decidido crear una clase que represente un resultado general independientemente del método usado llamado *ResultRest<T>*. En esta clase se alberga la lista de ítems devueltos de tipo T y un booleano indicando si hay más resultados.

El método *search/advanced* en concreto devuelve una lista de preguntas. Mediante el filtro explicado en el apartado anterior sabemos qué atributos debe tener la clase pregunta.

Todos sus atributos son primitivos exceptuando la lista de respuestas asociadas por lo que debemos diseñar también la clase respuesta.

La clase respuesta consta de 4 atributos primitivos vistos en el apartado anterior, eso significa que ya no tenemos que diseñar más clases adicionales.

Por último, para la creación de filtros se ha implementado la clase *Filter* con los atributos que indican los campos incluidos, el tipo y la cadena que representa dicho filtro para utilizarlo en futuras peticiones.

Un requisito fundamental para que funcione la integración completa con GSON es el uso de constructores vacíos en cada una de las clases y la existencia de los correspondientes *getters* y *setters* de cada atributo.

El diseño y las dependencias de cada clase se muestran en la figura 25 para visualizar mejor lo que se ha implementado.

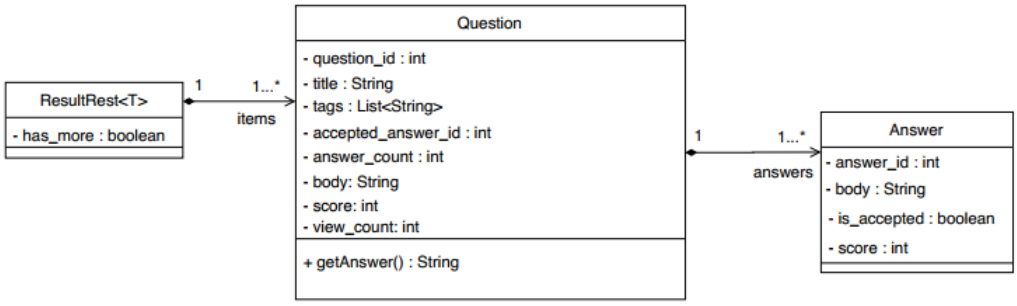

Figura 17. Diagrama de los modelos implementados para Stack Overflow

4.2 API Intra Stack Exchange

Una de las ideas más atractivas de este plugin es la posible incorporación de otros servicios distintos de Stack Overflow. Intra Stack Exchange es una propuesta del potencial del proyecto que consiste en un foro muy básico pero funcional y eficaz.

A través de ISE el usuario puede crear una pregunta con su descripción y añadir una respuesta si lo ve de utilidad. Una vez publicada aparecerá en el resultado de las posteriores búsquedas de otros usuarios. El uso de un servicio de ayuda interna proporciona una coherencia en el código y una personalización de las consultas relacionadas específicamente para el proyecto actual.

La implementación de Intra Stack Exchange se ha configurado mediante MongoDB, una base de datos no relacional y que proporciona varias características muy útiles, como la creación automática de la base de datos y la colección, uso de filtros con expresiones regulares etc.

El almacenamiento de las preguntas se hará en la base de datos llamada IntraseDB y más en concreto en la colección IntraSE. Dentro de esta colección se hallarán las preguntas cuyos campos son:

- question_id – Id único para cada pregunta.
- title – Cadena de texto que representa el título de la pregunta.
- body – Descripción detallada del problema a resolver.
- answer – Descripción de la solución a dicho problema.
- owner – Nombre del creador de la pregunta

4.2.1 Operaciones

Las principales funciones que debe realizar el plugin relacionado con este servicio son la inserción de nuevas preguntas y la búsqueda de las ya existentes en la base de datos. Debido a que estas funcionalidades se tienen que realizar a través de vistas en Eclipse, se ha usado un controlador para el manejo de MongoDB mediante código java.

Antes de empezar a usar la base de datos con este controlador hay que instanciar una conexión a través de un host y puerto. Con ello se obtiene un cliente con el que realizar consultas a un servidor corriendo en la máquina del host proporcionado. Usando el cliente obtendremos la base de datos IntraseDB y con ésta la colección IntraSE que mantendremos en un atributo privado llamado *collection*.

A la hora de **insertar nuevas preguntas** primero se comprueba si ya existe en la base de datos mediante su id. Para realizar dicha comprobación se hace uso del método *find* definido dentro de *collection* junto con un filtro pasado por argumento que comprueba el campo *question_id* con el id de la pregunta a insertar. En caso contrario se crea el documento con cada uno de los campos del objeto pregunta y se inserta en la colección.

Para la creación de documentos existe una clase llamada *Document* que representa un archivo BSON. El archivo BSON es la representación binaria de un archivo JSON y es el formato utilizado por la base de datos MongoDB. En el constructor de *Document* podremos incluir una clave de tipo String y un valor de tipo Object. El tipo del valor proporcionado debe ser compatible con los tipos que se definen en MongoDB. Llamando al método append() se puede añadir un par clave-valor y gracias a que este método devuelve un documento es posible seguir llamando a append() para seguir añadiendo claves y valores como se muestra en la figura 26.Por último, en el caso en que se quiera insertar un único documento se hará mediante el método insertOne() del objeto *collection*.

```
public void insertQuestionIntraSE(QuestionIntraSE question) {
    int question_id = question.getQuestion_id();

    // Check if question exists in database
    FindIterable<Document> iterable = collection.find(Filters.eq("question_id", question_id));
    if (iterable.first() == null) {
        Document document = new Document("question_id", question_id).append("title", question.getTitle())
                .append("body", question.getBody()).append("answer", question.getAnswer())
                .append("owner", question.getOwner());
        collection.insertOne(document);
    }
}
```

Figura 18. Código java para la inserción de una pregunta en una base de datos MongoDB

En la búsqueda de preguntas se usará el método *collection.find()* usado anteriormente para la comprobación de la existencia de una pregunta en la base de datos. En combinación con este método se utilizará un filtro de tipo expresión regular aplicado a uno de los campos de la pregunta, ya sea el título o el cuerpo.

El objetivo principal es que el usuario introduzca una cadena de caracteres, el cual es un conjunto de palabras separadas por espacios en blanco, y que todas las palabras introducidas estén en el campo a filtrar. El orden de las palabras no tiene importancia y no tienen por qué ser vecinas entre sí. Para conseguir este resultado se va a hacer uso del *lookahead* que proporciona las expresiones regulares.

Un *lookahead* tiene dos variantes: positivo o negativo. Ambos funcionan de manera opuesta, pero siguiendo el mismo principio. Su formato se muestra en la figura 27 y consta de dos partes representadas como *q* y *u.*

Figura 19. Lookahead negativo y positivo respectivamente

El lookahead negativo devuelve q cuando no está seguido por u, por ejemplo, en la cadena "juanpedroantonio" si lo evaluamos con la expresión regular "juan(?!antonio)" el resultado dará una coincidencia en juan ya que éste está seguido por pedro y no por Antonio.

Por otro lado el lookahead positivo devuelve q cuando está seguido por u, por ejemplo, en la cadena "juanpedroantonio" si esta vez lo evaluamos con su versión positiva "juan(?=pedro)" el resultado devolverá juan ya que está seguido de pedro.

Con el uso del lookahead positivo se ha llegado a la siguiente expresión regular teniendo como entrada la cadena "juan pedro antonio":

*^(?=.*juan)(?=.*pedro)(?=.*antonio)*

Como toda expresión regular se comienza el procesamiento de la cadena de izquierda a derecha. Lo primero que nos encontramos es el lookahead *^(?=.*juan)* que devolverá el principio de la cadena (^) si ésta va seguida de cualquier conjunto de caracteres (.*) y la palabra juan. Por tanto, dada la entrada "juan pedro antonio" este primer lookahead devolverá ^ y la expresión regular se simplificará en *^(?=.*pedro)(?=.*antonio)*. Como se puede observar el procesamiento ahora es el mismo que el anterior solo que con otra palabra. Cada lookahead de esta expresión irá devolviendo el principio de la cadena hasta llegar al último. Si el último devuelve el principio de la cadena se interpretará como que la entrada posee estas tres palabras. También, al usar el conjunto de caracteres (.*) junto con la palabra deseada se consigue que el orden de los lookahead en la expresión regular carezca de importancia.

Con todo esto, la entrada a procesar será el campo a evaluar, en este caso el título de la pregunta, y la expresión regular se creará de manera dinámica dividiendo la consulta proporcionada por el usuario por palabras que se transformarán en lookahead para cada una de ellas.

```
String[] words = query.split(" ");

// positive lookahead pattern to search
String pat = "^";
for (String word : words) {
    pat += "(?=.*" + word + ")";
}
System.out.println(pat);
FindIterable<Document> iterable = collection.find(Filters.regex("title", pat));
```

Figura 20. Código java para la búsqueda de preguntas en una base de datos MongoDB

La eliminación de una pregunta se realiza mediante el método *collection.deleteOne()* con un filtro para comprobar el id de la pregunta mientras que la actualización se consigue

llamando a *collection.updateOne()* y pasando por argumento la pregunta en forma de Document y comparando con un filtro su id.

Para concluir, todas estas operaciones están encapsuladas bajo la clase IntraSEProvider mostrada en la figura x, la cual sigue el patrón de diseño Singleton para otorgar un acceso global en el código.

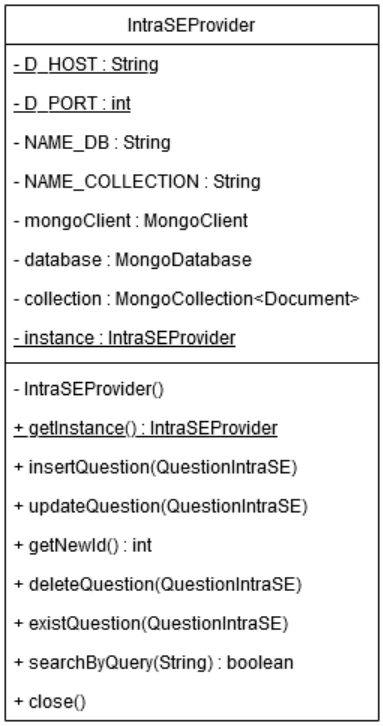

Figura 21. Clase IntraSEProvider con sus atributos y métodos

4.2.2 Modelo

Los datos intercambiados entre la base de datos y el plugin relacionados con el servicio Intra Stack Exchange se encuentran en la clase QuestionIntraSE la cual representa la pregunta con la respuesta asociada que el usuario crea y puede modificar. Además del cuerpo de la pregunta y la respuesta, también se incluye un id único, el título y el nombre del creador.

El método *toQuestionIntraSE()* se encarga de convertir un documento pasado por argumento a una instancia de tipo QuestionIntraSE. De manera opuesta se implementa toDocument() que se ocupa de crear un documento a partir de un objeto QuestionIntraSE. Estos métodos son estáticos para ser accedidos a través de la clase sin la necesidad de tener una instancia y resultan útiles a la hora de comunicarse con la base de datos.

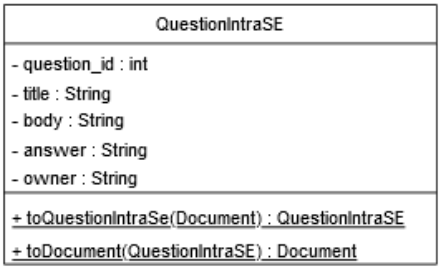

Figura 22. Clase QuestionIntraSE con sus atributos y métodos

4.3 Motor de búsqueda

En los apartados anteriores se ha explicado las dos fuentes principales de donde el plugin extraerá información para mostrársela al usuario. El objetivo de la búsqueda de preguntas es la unificación de resultados de ambas fuentes en un único espacio. Debido a este requisito surgen dos problemas: ¿Cuál es el orden de los resultados? ¿Cómo unificarlos si tienen diferentes características?

4.3.1 Orden de los resultados

En el plugin existen dos tipos de resultados: preguntas extraídas del foro Stack Overflow y preguntas extraídas de Intra Stack Exchange.

El primer tipo presenta una mayor cantidad de información así como una enorme diversidad de temas relacionados. Por ello, Stack Overflow es la principal fuente de donde el usuario debería encontrar la respuesta a su consulta.

Por otro lado, Intra Stack Exchange tiene como objetivo la creación de preguntas asociadas y personalizadas a un proyecto existente. Esto supone un mayor interés frente a un conjunto generalizado como lo es Stack Overflow.

Por último, ya que cada pregunta puede ser marcada como recomendada, antes de fusionar ambos resultados se ordenará cada uno de ellos de manera que los recomendados se ubiquen al principio de cada lista.

4.3.2 Unión de modelos de diferentes servicios

Cada servicio puede extraer información de diferentes fuentes (base de datos local, api REST, web…) y por tanto puede presentar diferentes campos no existentes en otros servicios. Esto aporta un enriquecimiento y diversidad al plugin así como una experiencia cómoda al usuario. Por el contrario, dicha diversidad supone una dificultad extra al unificar toda la información extraída.

Para manejar este escenario se propone la creación de interfaces que corresponden a funcionalidades a implementar si se quiere que la información aparezca unificada con el resto de los resultados. Por ejemplo, si una pregunta desea aparecer en la lista de búsqueda deberá implementar la interfaz IQuestionSearch.

Existen tres tipos de interfaces: IQuestionSearch, IQuestionRecommendation, ICodeSearch.

Los resultados de una consulta se presentan en una tabla con tres columnas. En concreto se muestra el título, la puntuación y el servicio de donde proviene la pregunta. Por tanto, IQuestionSearch contendrá tres métodos que retornarán cada uno el título, la puntuación y el origen. Además de estos tres métodos también se encuentra *showView()* que será llamado cuando el usuario pulse dos veces con el ratón sobre la pregunta en la tabla. De esta manera cada pregunta podrá implementar su propia interfaz para mostrar la información deseada.

Para que una pregunta pueda ser almacenada en la base de datos de recomendados debe implementar la interfaz de IQuestionRecomendation la cual obliga a proporcionar un id y devolver la colección a la que está asociada el servicio de la pregunta seleccionada. Este último método es necesario ya que la base de datos de recomendación creará una colección distinta por cada servicio existente. Esta decisión de diseño se debe a que dos o más servicios puedes proveer una pregunta con un mismo id y por tanto no saber qué servicio escoger para recuperar la pregunta.

Por último tenemos la interfaz ICodeSearch con la cual una pregunta podrá ser incluida en la función de la búsqueda de códigos. En este caso solamente se necesitará la respuesta asociada para poner en marcha el filtro que busca códigos y los muestra al usuario.

Las clases que implementan estas interfaces son contenedores o wrappers asociados a los modelos de las preguntas de cada servicio. Se identifican por llevar una W seguido del nombre del modelo al cual contienen. Un ejemplo de contenedor lo tenemos en la clase WQuestionIntraSE en la figura 31.

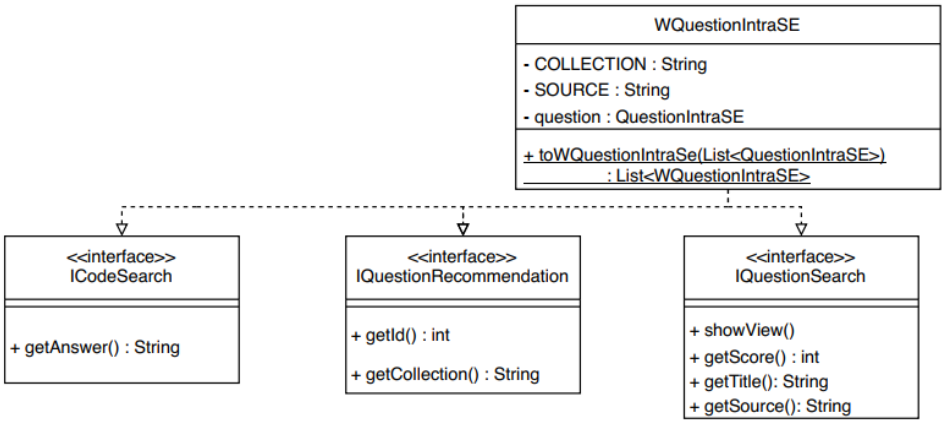

Figura 23. Clase WQuestionIntraSE con sus atributos e interfaces

Con el uso de estos contenedores ya seremos capaces de unificar todas las preguntas añadiéndolas a una lista de tipo IQuestionSearch y retornándola como devolución del método de búsqueda.

4.4 Vistas

Las vistas forman una parte fundamental de la integración del programa con el entorno de desarrollo Eclipse. Mediante estas interfaces gráficas el usuario puede usar de una manera interactiva e intuitiva las diferentes funcionalidades que hemos estado mencionando en este documento.

Esta sección va a estar dividida en 7 apartados. En el primer apartado se expondrá el uso de los puntos de extensión asociados a las vistas de Eclipse lo que nos permitirá integrar

nuestros diseños en el entorno de desarrollo. Los apartados del 2 al 5 coincidirán con cada una de las vistas a desarrollar, en concreto, la búsqueda de preguntas, presentación de la información de la pregunta seleccionada, creación de una pregunta Intra Stack Exchange y búsqueda de códigos. En cada uno de ellos se describirá el diseño de la interfaz gráfica y el comportamiento de cada uno de sus elementos actuadores (botones, tablas, entradas de texto…). El apartado 6 corresponderá a la incorporación de una nueva página en el menú de preferencias de Eclipse, ya que su punto de extensión no es igual que a la de las vistas se mostrarán las diferencias con estas. Como séptimo y último apartado se expondrá la manera de crear menús con los que mostrar las vistas de los apartados anteriores.

4.4.1 Puntos de extensión

Para poder insertar nuestras vistas personalizadas en Eclipse debemos extender el punto de extensión org.eclipse.ui.views . Dentro de este punto de extensión tendremos que definir un nodo llamado “category” el cual se encargará de agrupar diferentes vistas. También habrá que definir un nodo llamado “view” con el que se hará referencia a nuestra vista personalizada. Los atributos de “view” serán:

- id - id único que identifica a la vista.
- category - id de la categoría previamente creada.
- class - Clase java que hereda de la clase ViewPart conteniendo todas las creaciones de los elementos gráficos necesarios así como de sus comportamientos.
- icon - Icono mostrado al usuario en la parte superior izquierda del título de la vista.
- name - Título de la vista que se mostrará en la cabecera de la vista.

Un ejemplo de extensión de una vista en un plugin se muestra en el listado 1.

```
<plugin>
   <extension
         point="org.eclipse.ui.views">
      <category
            id="helpstack"
            name="HelpStack">
      </category>
      <view
            category="helpstack"
            class="helpstack.views.SearchView"
            icon="icons/sample.png"
            id="helpstack"
            name="HelpStack">
      </view>
  </extension>
</plugin>
```

Listado 1. Declaración de extensión del punto de extensión *org.eclipse.ui.views*

Cuando una vista va a ser creada y mostrada al usuario, el punto de extensión llamará al método *createPartControl(Composite parent)* definido en la clase ViewPart. En el listado 1 la clase *helpstack.view.SearchView* hereda de ViewPart y sobrescribe el método *createPartControl* para crear sus propios widgets SWT con sus correspondientes disposiciones. Todos estos elementos que crea la nueva clase deben ser añadidos al atributo parent para que puedan visualizarse en la vista resultante.

Todas las vistas que veremos a continuación tienen la misma estructura que la mostrada en el listado 1. El uso del patrón MVC (Modelo Vista Controlador) en las vistas hace que el diseño se implemente en la clase que hereda de ViewPart, más en concreto en el método *createPartControl*, y el comportamiento en su controlador asociado.

4.4.2 Búsqueda de preguntas

Ya hemos visto como obtener preguntas de los servicios Stack Overflow e Intra Stackexchange así como ordenarlos y priorizarlos junto el sistema de recomendados. También vimos la implementación del motor de búsqueda cuya función era proveer el método de búsqueda de preguntas para que pudiera ser accedió en cualquier parte del código.

Ahora nos centraremos en cómo poner a disposición del usuario todos los elementos necesarios para que pueda realizar el proceso de consulta de manera cómoda y priorizando los objetivos de Helpstack.

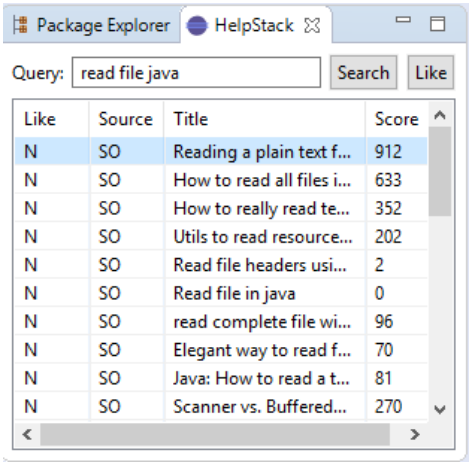

Figura 24. Vista SearchView para la consulta de preguntas.

4.4.2.1 Diseño

Uno de los objetivos importantes de este proyecto era evitar descontextualizar al usuario. Por regla general cuando un programador consulta una duda en la web debe salir del IDE, lo que supone apartar el foco de atención en el editor.

Como solución a este problema, la vista encargada de realizar consultas aparecerá al lado de una de las vistas de apoyo como puede ser la consola o el explorador del proyecto. Con esto conseguimos que permanezca visible el editor mientras el usuario se encuentre en el proceso de consulta.

Otro punto fundamental del plugin es la de evitar interfaces con muchas opciones o sobrecargar una vista con muchas funcionalidades para que la curva de aprendizaje sea la menor posible. Para lograr un proceso rápido y sencillo lo mejor es proporcionar al usuario un espacio para que introduzca su consulta, un botón para realizar dicha consulta y una tabla mostrando los resultados como se observa en la figura 32. Para poder visualizar el contenido de la pregunta se deberá realizar una doble selección sobre la entrada de la tabla deseada.

El usuario también tiene la posibilidad de marcar a una pregunta con un “me gusta” con la finalidad de que si volviera a usar una consulta parecida o igual, esa pregunta se mostraría la primera en la tabla.

La entrada de texto y los botones de buscar y “me gusta” son widgets SWT los cuales están asociados a un GridLayout. Por el contrario, la tabla que muestra los resultados es un Viewer. Un Viewer es una clase que une una vista con un modelo. En este caso se ha usado TableViewer que, en definitiva, es un widget Table asociado con un array de tipo genérico.

Al instanciar un TableViewer hay que configurar e inicializar algunos parámetros importantes para que funcione de la manera deseada. Uno de estos parámetros es asociar a la tabla u proveedor de contenido o ContentProvider, que en nuestro caso será la clase ya existente ArrayContentProvider, el cual permite manejar un array de objetos como modelo.

El segundo paso es la creación de las columnas instanciando TableViewerColumn por cada una de ellas e inicializando sus títulos y otras propiedades. Una vez creada la columna habrá que asociarla un proveedor de información o LabelProvider, el cual se encarga de decidir qué mostrar en esa columna dado un objeto proporcionado por el proveedor de contenido. Existe un proveedor llamado ColumnLabelProvider que mediante la sobreescritura de su método *getText()* se devolverá la información correspondiente a esa columna dado un objecto.

4.4.2.2 Comportamiento

La implementación de las acciones de los elementos gráficos será delegada al método *registerListeners* de la clase SearchController. Este método recibe la clase de la vista SearchView como parámetro para poder acceder a los elementos SWT y manipularlos.

Cuando se selecciona el botón de búsqueda primero se valida la entrada de texto. En caso de estar vacía la entrada de texto no se mostrará nada y terminará el algoritmo. En caso contrario se procederá a recuperar las preguntas que coincidan con la consulta del usuario mediante el motor de búsqueda explicado en apartados anteriores. El método *serachQuestion()* devolverá un array con las preguntas el cual se establecerá como entrada de la tabla mediante su método *setInput()*. En caso de hubiera un error en el proceso de búsqueda se capturaría la excepción y mostraría un mensaje de error al usuario.

Al realizar una sola selección sobre una pregunta el estado del botón “me gusta” debe cambiar dependiendo de si esa pregunta existe o no en la base de datos de recomendados. Por ello, si existe la pregunta en dicha base de datos el botón cambiará al estado “no me gusta” y por el contrario, si no exista en la base de datos el botón cambiará al estado “me gusta”. Si no hay ninguna pregunta seleccionada se deshabilitará el botón dejándolo inaccesible pero visible al usuario.

En el caso del botón “me gusta” se obtendría la pregunta que está seleccionada en la tabla y se comprobaría su existencia en la base de datos de recomendados. En caso de existir da a entender que el botón se encuentra en el estado de “no me gusta” y por tanto se eliminaría de la base de datos, acto seguido cambiaría el botón al estado “me gusta”. En caso de que no existiera en la base de datos se insertaría y se cambiaría al estado “no me gusta”. El objetivo de este comportamiento es obtener funcionalidades contrarias en un mismo botón y no tener que crear dos botones.

El usuario puede realizar una doble selección sobre la pregunta, lo que provocará que se recupere la pregunta seleccionada y se llame a la función *showView()* definida gracias a la interfaz *IQuestionSearch*.

4.4.3 Información de la pregunta

Una vez realizado el proceso de búsqueda y escogida la pregunta que parece más adecuada a nuestra consulta comienza el proceso de inspección a través de los campos de la pregunta. En este apartado se mostrará la distribución de los campos asociados a la pregunta de Stack Overflow (figura 33) e Intra Stackexchange (figura 34).

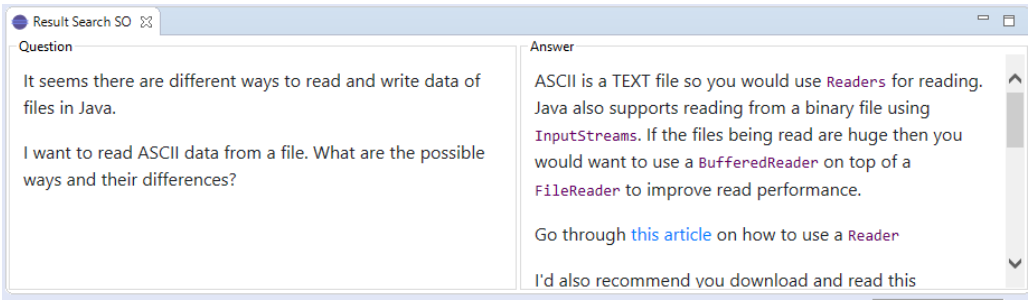

Figura 25. Vista ResultSearchSOView mostrando la información de una pregunta de Stack Overflow.

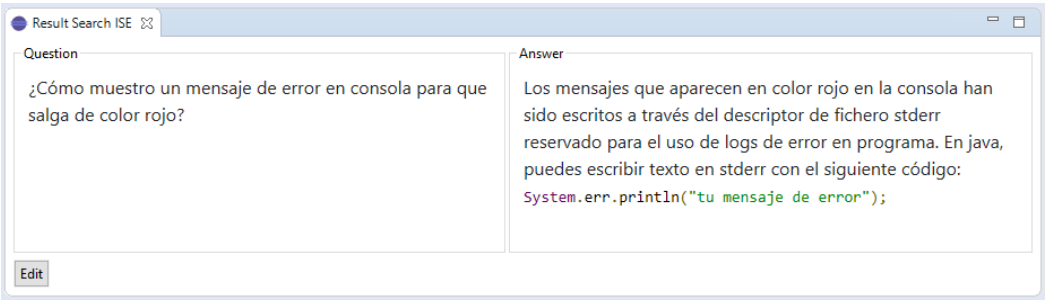

Figura 26. Vista ResultSearchISEView mostrando algunos campos de una pregunta de origen ISE.

4.4.3.1 Diseño

Durante el proceso de consulta el programador busca una respuesta aceptada por el usuario que creó la pregunta. Por ello se ha decidido mostrar únicamente el enunciado de la pregunta, para que el programador compruebe que tiene un problema parecido, y su respuesta asociada. De tal manera la visualización queda limpia y concisa para una rápida identificación de los elementos.

En cuanto a mostrar el texto de los campos se ha optado por usar el widget Browser el cual permite visualizar un documento HTML con su estilo asociado además de poder navegar por los diferentes enlaces que aparecen en los cuerpos de las preguntas y respuestas. La elección de HTML como representación de la información supone una ventaja ya que los campos de las preguntas que devuelve la API REST de Stack Overflow también usan este formato.

Todo código presente en un documento HTML viene envuelto por las etiquetas <code></code> con lo que puede ser usado para otorgar un estilo basado en el uso de colores para una lectura más fluida y familiar semejándose a un editor. Para ello se hace uso de la librería *Prettify* desarrollado por Google. Con esta librería incluida en un documento HTML se puede asociar la clase *prettyprint* a un elemento y automáticamente la librería aplicará un estilo genérico al contenido de ese elemento.

4.4.3.2 Comportamiento

La creación de esta vista comienza cuando el usuario realiza una doble selección sobre una entrada en la tabla de búsqueda. En ese momento se recupera la pregunta seleccionada y se llama a su método *showView()*. Dentro de este método se obtendrá una instancia de la vista deseada dependiendo del tipo de pregunta y se llamará al método updateView() implementada en la propia vista pasando como argumento la pregunta.

En este método es donde se inicializará el contenido de los diferentes widgets SWT que así lo necesiten. En nuestro caso hay que insertar el enunciado de la pregunta así como el contenido de la respuesta. No obstante, antes se han de aplicar algunos cambios al contenido de ambos campos para poder mostrar los estilos mencionados en el apartado de diseño.

El primer paso para realizar estos cambios es añadir la clase *prettyprint* a los elementos con etiquetas <code></code> mediante el uso de un analizador HTML como Jsoup. Después hay que insertar este contenido en el cuerpo de una plantilla HTML en cuya cabecera se ha incluido la librería *Prettify* y *Bootstrap* para garantizar un estilo uniforme y moderno. De esta manera la lectura de la pregunta y la respuesta a se presenta accesible y cómoda a la vista del programador como se muestra en las figuras 32 y 33.

4.4.4 Creación de preguntas

En este apartado se va a bordar el proceso de creación de una pregunta perteneciente a Intra Stackexchange y sus decisiones de diseño. Un ejemplo de esta vista se muestra en la figura 35.

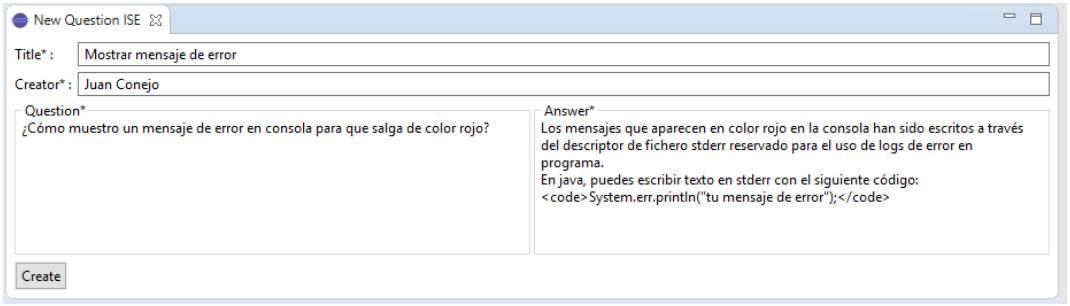

Figura 27. Vista NewQuestionISEView para la creación de nuevas preguntas en Intra Stack Exchange.

4.4.4.1 Diseño

La vista encargada de crear e insertar preguntas pertenecientes a Intra StackExchange se compone de cuatro entradas de texto y un botón. Estas cuatro entradas se corresponden al título, nombre del creador, enunciado de la pregunta y el contenido de la respuesta. Debido a que el texto de la pregunta y la respuesta van a estar formateadas en HTML, cuando el usuario quiera indicar un bloque de código deberá contenerlo con la etiqueta <code></code>.

4.4.4.2 Comportamiento

El usuario deberá terminar el proceso seleccionando el botón *"Create"* con el que dará comienzo la fase de validación e inserción en la base de datos. En la fase de validación se comprobará que ningún campo quede vacío y que en los campos de pregunta y respuesta no se presente ninguna etiqueta diferente a <code>. La validación de los campos es una tarea

importante sobre todo si el contenido se desea introducir en un formato HTML y mostrar en un navegador, ya que un atacante podría introducir código malicioso escrito en lenguaje de cliente web como por ejemplo Javascript. En el proceso de inserción se creará la pregunta, se insertará en la base de datos mostrando un mensaje informativo al usuario y por último vaciando los campos de la vista para una nueva creación si se desea.

4.4.5 Búsqueda de códigos

En ocasiones el programador busca un uso de un método o algoritmo específico y no da importancia al texto asociado a la respuesta. Es muy común navegar por diferentes preguntas hasta encontrar la manera en la que al usuario le gusta implementar su funcionalidad. Por ello en el plugin, de manera alternativa a la búsqueda común, se proporciona un diálogo en donde insertar una consulta, mostrar los códigos contenidos en el cuerpo de las preguntas en una tabla, y estilizarlos para una fácil lectura en la parte inferior.

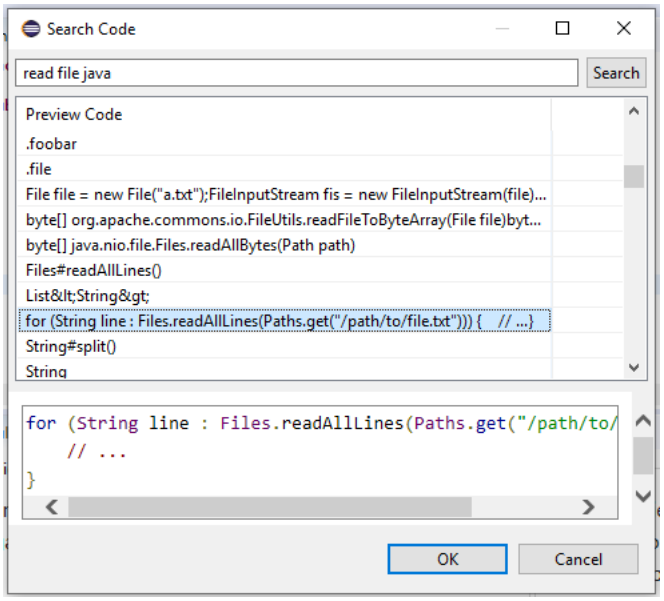

Figura 28. Vista SearchCodeDialog para la búsqueda de códigos de la consulta solicitada.

4.4.5.1 Diseño

El diseño consiste en una entrada de texto y un botón para realizar una consulta. Nuevamente se usa el Viewer TableViewer para mostrar de manera organizada las vistas previas de los resultados. Por último se utilizará un navegador WidgetBrowser para usar la misma técnica de estilo que al mostrar la información de una pregunta en una búsqueda normal. A diferencia del resto, esta vista es un cuadro de diálogo y por tanto se mostrará encima de todas las vistas de Eclipse como se muestra en la figura 36.

4.4.5.2 Comportamiento

La clase SearchCodeController se encarga de implementar el comportamiento de los botones y la tabla teniendo en cuenta el resto de los elementos gráficos de la vista. Al seleccionar el botón de búsqueda el controlador comprobará si la entrada de texto está vacía. En caso contrario realizará una llamada a un método del motor de búsqueda encargado de encontrar códigos dado una consulta.

En concreto este método obtendrá las preguntas resultantes de la consulta como si de una búsqueda normal se tratara para posteriormente parsear todas las respuestas y recuperar solo el texto contenido en las etiquetas <code></ code>. Una vez obtenido todos los códigos, se insertarían en la tabla.

El usuario podrá seleccionar una entrada de la tabla lo que mostrará en detalle el contenido del código estilizado mediante la librería *Prettify* de Google. También el usuario podrá usar las flechas del teclado para avanzar o retroceder en la tabla con lo que se actualizará la visualización del contenido.

4.4.6 Preferencias

La base de datos Intra StackExchange por defecto se ubica en la maquina donde se usa el entorno Eclipse pero se puede indicar la IP y el puerto si se desea que la base de datos esté instalada en otro servidor. El hecho de poder elegir en qué base de datos consultar las preguntas puede ser muy útil ya que permitiría tener una fuente de información común entre varios equipos de desarrollo. En el ejemplo de una empresa que esté desarrollando un proyecto y quisiera dejar documentación o resolver las dudas iniciales de los nuevos programadores podría hacer uso de este plugin almacenando todas esas preguntas en una base de datos común.

Para que sea más intuitivo y estandarizado se ha optado por incluir un menú en las preferencias de Eclipse con el que poder configurar las direcciones IP y puertos de las dos bases de datos utilizados por el plugin. En la implementación de este menú es necesario extender el punto de extensión *org.eclipse.ui.preferencePages* en el cual referenciaremos la clase encargada de definir dónde se almacenan las preferencias, la descripción del menú y los diferentes campos.

En un principio estos campos deberían tener un valor por defecto. Para ello se necesita extender otro punto de extensión llamado *org.eclipse.core.runtime.preferences* y referenciar nuevamente a la clase encargada de inicializar los valores por defecto. El lugar donde se van a almacenar estas preferencias iniciales proviene del activador el cual posee un almacén de preferencias por defecto. Seguidamente para crear cada una de las preferencias se inserta en el almacén un par clave-valor donde la clave será una cadena identificando a la preferencia y el valor será el contenido inicial.

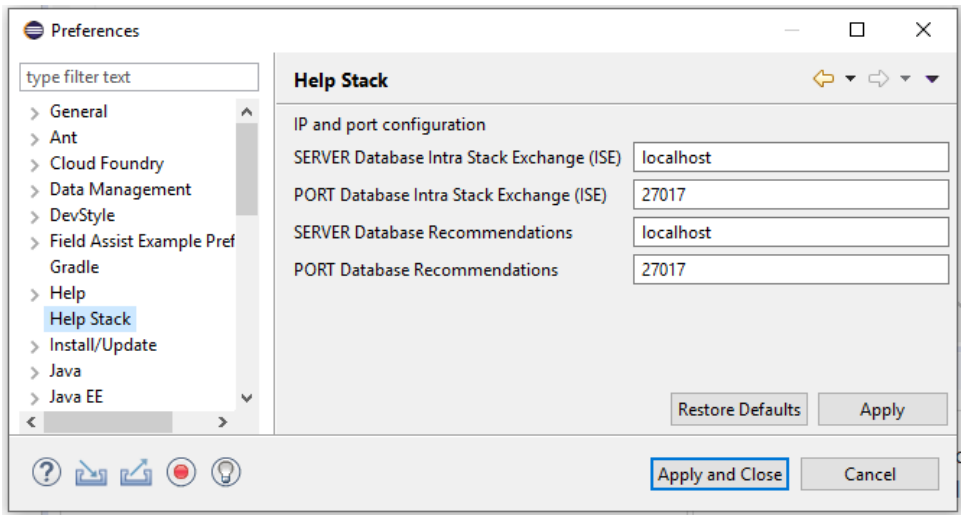

Figura 29. Preferencias del plugin añadidas al menú de preferencias de Eclipse.

4.4.6.1 Diseño

Una página de preferencias funciona de manera distinta a una vista, de manera que ya no se necesita usar elementos SWT sino que existen por defecto métodos para añadir campos con tipos concretos de datos. En este caso se añadirán cuatro campos de tipo *StringFieldEditor*, es decir, campos de tipo cadena de caracteres a los que se asocian con las preferencias definidas en el inicializador y se definen sus cabeceras descriptivas.

4.4.6.2 Comportamiento

Al tratare de una página de preferencias, por defecto se generan dos botones: restaurar y aplicar. Los comportamientos de ambos botones se implementan en los métodos *performDefaults()* y *performOk()* respectivamente. En el caso de restaurar simplemente se actualiza la base de datos y se vuelve a llamar a *performDefaults()* de la clase padre ya que los valores por defecto ya están definidos previamente. Por otra parte, en el caso de aplicar, simplemente se actualizarán las bases de datos en las cuales se leerán nuevamente las preferencias y se crearán otras instancias con las configuraciones modificadas.

4.4.7 Menús y Atajos

Todas las vistas que hemos estado viendo podemos mostrarlas en el entorno de desarrollo mediante Window→Show View→Other→HelpStack debido a que hemos extendido cada una de ellas del punto de extensión *org.eclipse.ui.views*. De manera alternativa se ha implementado un menú con tres entradas para mostrar la vista de búsqueda, crear nueva pregunta en ISE y búsqueda de códigos.

Para ello lo primero que se debe hacer es crear un comando para cada uno de ellos extendiendo de *org.eclipse.ui.commands* y asociándolos con una categoría. Una vez definido los comandos hay que crear los manejadores extendiendo de *org.eclipse.ui.handlers*, los cuales relacionan un comando con una clase Java. Esta clase Java será la encargada de implementar la funcionalidad deseada al ejecutar el comando. Finalmente para crear visualmente el menú con sus entradas se debe extender de *org.eclipse.ui.menus*. Para este punto de extensión hay que definir primero el menú, en donde uno de sus atributos es el nombre, y dentro del menú deben estar definidas las entradas, las cuales relacionan un comando con un id de entrada.

Por lo tanto, cada entrada del menú hace referencia a un comando y un manejador relaciona un comando con una clase Java la cual aportará cierta funcionalidad, en nuestro caso mostrar una vista dependiendo del comando de origen.

El uso de atajos ofrece una mayor comodidad a la hora de realizar una acción repetida como puede ser copiar y pegar o en nuestro caso mostrar la vista de búsqueda para realizar una consulta. Para crear un atajo se tiene que extender el punto de extensión *org.eclipse.ui.bindings* e incluir una clave por cada atajo que se quiera realizar. Una clave relaciona un comando con una secuencia de teclas. Para mostrar la vista de búsqueda se ha optado por la secuencia Ctr+Alt+Q y para el diálogo de búsqueda de códigos la secuencia Ctrl+Alt+C. De esta manera se consigue que, usando el atajo, escribir la consulta y pulsar “enter” en el teclado sea una manera rápida de obtener resultados.

4.5 Despliegue y exportación

Una vez finalizado el desarrollo se procede a exportar el plugin para poder ser desplegado en otros entornos de desarrollo. Para ello hay que tener en cuenta ciertas propiedades a la hora de empaquetar todo el proyecto. Estas propiedades se definen dentro del archivo *build.properties* y algunas de ellas son:

- source – Indica la ruta donde se ubica el código fuente del plugin
- output – Indica la ruta donde se ubica los binarios generados al compilar el plugin.
- bin.includes – Binarios y recursos externos al plugin que deben ser incluidos en el empaquetado final.
- src.includes – Códigos fuentes externos al plugin que deben ser incluidos en el empaquetado final.

Un ejemplo de archivo de configuración se muestra en el listado 2.

```
source.. = src/
output.. = bin/
bin.includes = .,\
               META-INF/,\
               plugin.xml,\
               libs/gson-1.4.jar,\
               libs/mongo-java-driver-3.10.2.jar,\
               libs/ini4j-0.5.4.jar,\
               libs/jsoup-1.12.1.jar,\
               template/,\
               libs/httpclient-4.5.10.jar,\
               libs/httpcore-4.4.12.jar,\
               libs/commons-logging-1.2.jar
```

Listado 2. Archivo *build.properties* propio del plugin Helpstack.

También, en la pestaña *Runtime* del asistente ofrecido por PDE hay que incluir en el campo *Classpath* los archivos JAR utilizados como librerías externas.

Para exportar uno o varios plugins Eclipse dispone la idea de *feature*. Este concepto es realmente un proyecto nuevo de tipo Feature Project en el cual se especifica la información adicional en el proceso de exportación. En el proyecto, entre otras cosas, se define el nombre, id y versión de la *feature*. Además, se deben incluir los plugins que contiene dicho proyecto. Finalmente, se hace uso de la herramienta de exportación de la que dispone para empaquetarlo todo en un archivo ZIP.

Si se desea importar el plugin habrá que dirigirse al menú Help → Install New Software → Add… → Archive… y seleccionar el archivo ZIP resultante de la exportación. Otra manera de instalar HelpStack es mediante Help → Install New Software → Add… e introduciendo la dirección del sitio [11] del plugin. El código del proyecto es abierto y puede ser descargado [12] para añadir nuevas funcionalidades o modificaciones a las existentes.

5 Integración, pruebas y resultados

Una vez creada la herramienta de consulta se procederá a evaluarla siguiendo la escala de usabilidad System Usability Scale [13] o SUS. Esta escala se enfoca en medir la usabilidad de cualquier herramienta orientada al usuario mediante un formulario de 10 preguntas. El usuario puede evaluar cada pregunta del 1 al 5 significando la puntuación más baja estar totalmente en desacuerdo y la puntuación más alta totalmente de acuerdo. Las preguntas impares representan un aspecto positivo hacia la herramienta y las pares un aspecto negativo. Con este orden se obtiene una redundancia que evita las respuestas aleatorias y confirma las conclusiones. Para interpretar los resultados obtenidos se procede a escalar la puntuación de 0 a 100. Este proceso consiste en realizar los siguientes pasos en orden:

- Restar uno a las preguntas impares
- Restar desde cinco a las preguntas pares para que al sumar todas las preguntas no penalicen estas.
- Con los dos pasos anteriores todos los valores se escalan de 0 a 4 siendo la puntuación más alta la más positiva.
- Por último hay que sumar todos los valores de todas las preguntas y el resultado multiplicarlo por 2,5. Con esto obtendremos una puntuación comprendida entre 0 y 100.

Gracias a este proceso obtenemos un resultado comprendido de 0 a 100 pero seguimos sin saber que puntuación corresponde a una buena usabilidad y a una mala usabilidad. La única manera de averiguar esos valores es obtener un conjunto grande de resultados y realizar un rango de percentil. Ya que System Usability Scale se ha usado y se ha investigado durante muchos años se han hecho múltiples rangos de percentil y se ha llegado a una conclusión mostrada en la figura 38.

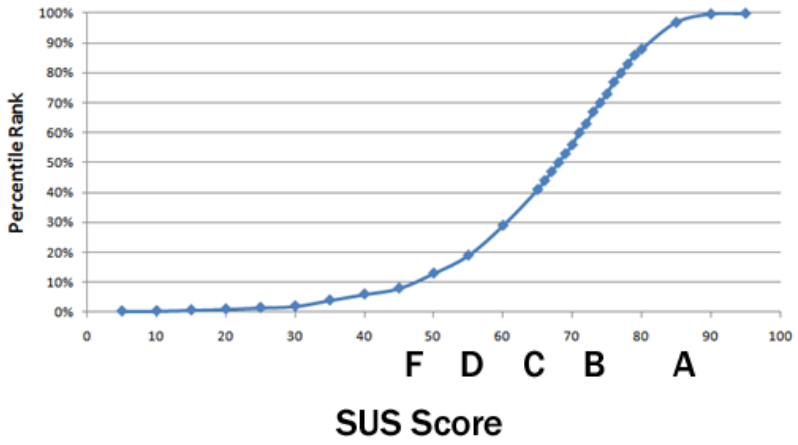

Figura 30. Rango de percentil asociado a la puntuación obtenida por SUS [14].

Como observamos en la figura 38, para obtener un porcentaje de 50 deberíamos teneruna puntuacion de 68 en SUS aproximadamente. Con este nuevo rango de valores se definen varias categorias referenciadas con las letras F, D, C, B y A siendo F la menos usable y siendo A la más usable.

Para realizar nuestra prueba también hemos añadido preguntas relacionadas más especificamente con nuestra herramienta con las que obtener más interpretación del contexto. El formulario elaborado puede ser consultado en el Anexo A.

Los participantes serán seis estudiantes de entre 23 y 25 años, todos graduados o en el último año de carrera de Ingeniería Informática que, antes de contestar las preguntas de SUS, deberán realizar un ejercicio que consistirá en crear un sencillo programa en Java usando el entorno de desarrollo Eclipse. La única restricción es la prohibición de uso de programas externos a Eclipse o HelpStack, obligando así a que los usuarios usen la herramienta cuando les surjan dudas sobre la realización del ejercicio. El ejercicio en cuestión puede ser consultado en el Anexo B. En las preguntas de contexto se observa que los participantes usan frecuentemente StackOverflow y algunos de ellos usan java y eclipse y otros no.

Los resultados de las preguntas asociadas a SUS se pueden observar en la figura 39. Cada color corresponde con la calificación de un participante en la pregunta asociada. Acumulando cada color se obtiene la puntuación total obtenida para esa pregunta. Los resultados concuerdan en que las preguntas impares son mucho menores que las pares afirmando así que los aspectos negativos tienen puntuaciones negativas y los aspectos positivos puntuaciones positivas. Para comparar la usabilidad de esta herramienta con la figura 38 debemos calcular su puntuación SUS siguiendo el proceso descrito anteriormente. La puntuación SUS obtenida de media es de 83,75 lo que sitúa a HelpStack en un 90% de usabilidad y catalogándola de tipo A.

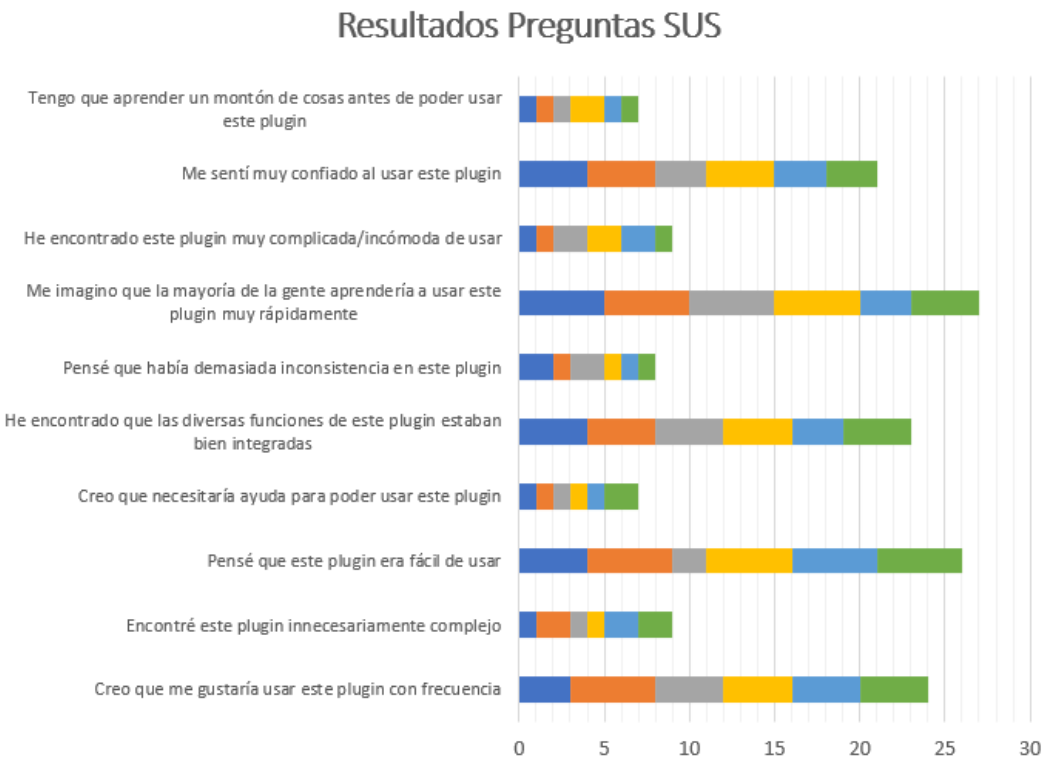

Figura 31. Gráfica de barras apiladas con los resultados de las preguntas asociadas a SUS.

En el caso de las preguntas asociadas al contexto pasa algo parecido, las impares en comparación a las pares tienen una puntuación más baja. Este patrón ocurre en todas excepto para la pregunta número cinco como se muestra en la figura 40. Esta pregunta está referida al uso de un navegador en contraposición a HelpStack. Por lo que podemos concluir, los usuarios, aun valorando de manera positiva al plugin, siguen prefiriendo un navegador para realizar consultas.

Figura 32. Gráfica de barras apiladas con los resultados de las preguntas asociadas al contexto.

En la sección de comentarios positivos los usuarios destacaron la utilidad del sistema de recomendación, la buena integración e interfaz limpia de HelpStack y la introducción al uso de Intra StackExchange con sus posibilidades de personalización de preguntas propias. Como aspecto negativo destacaron la incómoda introducción de código en las preguntas de Intra StackExchange así como pequeños errores visuales y mejoras en relación con el idioma.

Por último, la validez de estos resultados puede haber sido amenazados por un número bajo de participantes y por tanto sería necesario un conjunto mayor de usuarios para confirmar los resultados de la usabilidad.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

Tras haber visto las diferentes tecnologías utilizadas, la implementación del plugin y haber realizado pruebas de usabilidad podemos deducir varias conclusiones. HelpStack es un plugin que logra su objetivo: encontrar la solución de las consultas realizadas de una manera cómoda e integrada con el entorno de desarrollo.

La personalización de preguntas para la documentación de un proyecto ha tenido un impacto positivo en el grupo de usuarios que han podido probar la herramienta. Aunque la inserción de código dentro del cuerpo de las preguntas es mejorable, los usuarios vieron potencial en su uso empresarial, y más en concreto a la formación de nuevos programadores en un proyecto grande. El sistema de recomendados cumple con su cometido, recomendar aquellas preguntas que más le pueden servir al usuario, sobre todo en consultas parecidas a las previamente realizadas. La búsqueda de códigos resultó muy útil para evitar leer el contexto y buscar manualmente el código deseado, pudiendo así filtrar de manera rápida si el resultado interesa o no interesa.

En definitiva, proceso de implementación muestra la facilidad de incorporar nueva funcionalidad al entorno Eclipse junto con los servicios API REST y las pruebas determinan que las herramientas de consulta externas ejecutadas dentro de Eclipse ayudan con un rápido encuentro con la respuesta deseada.

6.2 Trabajo futuro

Aunque los resultados de HelpStack han sido positivos, la herramienta puede ser mejorable hablando en términos integración con el entorno de desarrollo. En concreto se podría:

- Determinar el lenguaje de programación el que está asociado el proyecto y añadir la etiqueta pertinente a los parámetros de búsqueda.
- Asociar preguntas a líneas de código en el editor para poder ser accedidas de manera más visual ayudando así con la documentación del proyecto.
- Incorporación de más foros de consulta a parte de Stack Overflow a través de extender puntos de extensión creados por HelpStack.
- En general, mejorar visualmente la interfaz gráfica con estilos más adecuados a los estándares.

Referencias

[1] «Osgi» , https://www.osgi.org/developer/architecture/ . [Último acceso: febrero 2020].

[2] «eclipsezone» , https://www.eclipsezone.com//articles/extensions-vs-services/. [Último acceso: febrero 2020].

[3] M. Scarpino, S. Holder, S. Ng y L. Mihalkovic, SWT/JFace in action, 2005, pp. 19-40.

[4] L. Ponzanelli, A. Bacchelli y M. Lanza, Seahawk: stack overflow in the IDE, ICSE, 2013, pp. 1295-1298.

[5] B. Angus Campbell y C. Treude, NLP2Code: Code Snippet Content Assist via Natural Language Tasks, ICSME, 2017, pp. 628-632.

[6] G. E. Krasner y S. T. Pope, A cookbook for using the model-view controller user interface paradigm in Smalltalk-80. J. Object Oriented Program, 1988, pp. 26-49.

[7] E. Gamma, R. Helm, R. Johnson y J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison Wesley Professional Comuting Series, 1994, p. 139.

[8] «Cloud Rest Api Client» , https://github.com/SAP/cloud-rest-api-client. [Último acceso: febrero 2020].

[9] «Stackoverflow java sdk» , https://github.com/sanjivsingh/stackoverflow-java-sdk. [Último acceso: febrero 2020].

[10] «Jaxrs» , https://github.com/Scalified/rest. [Último acceso: febrero 2020].

[11] «Update Site HelpStack» , https://jncho.github.io/site.xml. [Último acceso: febrero 2020].

[12] «HelpStack» , https://github.com/jncho/stackoverflow-eclipse/. [Último acceso: febrero 2020].

[13] J. Brooke, SUS: A “quick and dirty” usability scale, P. W. Jordan, B. Thomas, B. A. Weerdmeester y A. L. McClelland, Edits., Usability Evaluation in Industry, 1996.

[14] «measuringu» , https://measuringu.com/sus/. [Último acceso: febrero 2020].

Glosario

API	Application Programming Interface
Plugin	Aplicación que agrega funcionalidad a otra
API REST	API Representational State Transfer
Librería	Conjunto de funcionalidades escritas en un lenguaje de programación
Java	Lenguaje de programación orientada a objetos
IDE	Integrated Development Environment
OSGi	Open Services Gateway initiative
Equinox	Implementación certificada de la especificación OSGi
XML	eXtensible Markup Language
SWT	Standard Widget Toolkit
GUI	Graphical User Interface
IP	Internet Protocol
HTTP	HyperText Transfer Protocol
GET	Método definido dentro de HTTP
SQL	Structured Query Language
HTML	HyperText Markup Language
JSON	JavaScript Object Notation
MVC	Modelo Vista Controlador
BSON	Binary JSON
Id	Identificador
Parsear	Procesar y analizar una serie de símbolos para determinar su estructura
SUS	System Usabilty Scale

Anexos

A Formulario System Usability Scale

Preguntas generales: Marque la casilla correspondiente para cada una de las preguntas.

Indique su grado de uso del entorno de desarrollo Eclipse (1-poco/ninguno, 5-intensivo)	1	2	3	4	5
Indique la regularidad con la que programa en lenguaje Java (1-muy de vez en cuando, 5-a menudo)	1	2	3	4	5
¿Con qué frecuencia realiza consultas en el foro Stack Overflow? (1-muy de vez en cuando, 5-a menudo)	1	2	3	4	5

Instrucciones: Para cada afirmación, marque la casilla que mejor describa sus reacciones a la herramienta.

	totalmente de acuerdo → ← totalmente en desacuerdo				
Creo que me gustaría usar este plugin con frecuencia	1	2	3	4	5
Encontré este plugin innecesariamente complejo	1	2	3	4	5
Pensé que este plugin era fácil de usar	1	2	3	4	5
Creo que necesitaría ayuda para poder usar este plugin	1	2	3	4	5
He encontrado que las diversas funciones de este plugin estaban bien integradas	1	2	3	4	5
Pensé que había demasiada inconsistencia en este plugin	1	2	3	4	5
Me imagino que la mayoría de la gente aprendería a usar este plugin muy rápidamente	1	2	3	4	5
He encontrado este plugin muy complicada/incómoda de usar	1	2	3	4	5
Me sentí muy confiado al usar este plugin	1	2	3	4	5
Tengo que aprender un montón de cosas antes de poder usar este plugin	1	2	3	4	5
La información mostrada para cada pregunta es suficiente para identificar lo que se busca	1	2	3	4	5
Me he sentido incómodo teniendo el editor y las ventanas de consulta siempre visibles	1	2	3	4	5
He encontrado la solución a mi consulta de manera relativamente rápida	1	2	3	4	5
Prefiero usar un navegador aparte del entorno de desarrollo para realizar consultas	1	2	3	4	5
La búsqueda de códigos me ha servido para encontrar lo que buscaba	1	2	3	4	5
No veo la utilidad al sistema de recomendados para ninguna situación	1	2	3	4	5
Creo que Intra Stack Exchange podría ser útil durante el desarrollo o documentación de un proyecto ...	1	2	3	4	5
El uso de atajos del teclado para mostrar las ventanas de consulta me ha resultado incómodo	1	2	3	4	5

Por favor, indique tres aspectos positivos que desearía resaltar sobre la herramienta:

_____
_____
_____
_____
_____

Por favor, indique tres aspectos negativos que desharía resaltar sobre la herramienta:

_____
_____
_____
_____
_____

B Ejercicio para el uso de HelpStack

HelpStack: Plugin de consulta para Eclipse

Ejercicio

Introducción

Para probar la funcionalidad y la usabilidad de la herramienta se propone que el usuario desarrolle un programa sencillo en Eclipse en lenguaje Java. La restricción principal en este ejercicio es la prohibición en el uso de programas externos para la consulta de dudas como pueden ser el navegador o manuales externos. Cuando el usuario requiera de esta necesidad deberá hacer uso del plugin HelpStack para encontrar la solución a sus problemas.

Descripción del programa a desarrollar

Se pide al usuario que desarrolle un programa que cumpla las siguientes etapas:

- Pedir por consola el nombre del usuario
- Obtener la fecha actual
- Guardar ambos campos en un fichero con extensión .dat separados por el carácter '|'
- Leer el fichero y recuperar ambos campos
- Mostrar un mensaje al usuario con el formato "En el mes <mes> del año <año> el usuario <nombre del usuario> decidió que era un buen momento para recordar su entrada a este diario en el día <día>."

Conclusión

Una vez desarrollado dicho programa valore el uso de esta herramienta rellenando el formulario que se proporciona adjunto a este documento con nombre "HelpStack SUS".

C Explicación detallada de OSGi y Equinox

Para comprender el funcionamiento de un plugin dentro del programa Eclipse necesitamos primero tener una idea general de cómo está estructurado este entorno.

En cierto punto del desarrollo de Eclipse, los desarrolladores decidieron implementar una estructura modular debido a las enormes dimensiones del proyecto y para soportar la extensión del software mediante estos módulos. Existían varias especificaciones que garantizaban este objetivo y escogieron implementar un sistema **OSGi**.

OSGi es una especificación software, lo que significa que documenta una estructura necesaria para que se cumplan ciertos criterios. El objetivo de esta especificación es la creación de un programa que esté compuesto por múltiples **bundles** que se comunican a través de **servicios**. En la figura 1 se puede observar las capas que conforman OSGi.

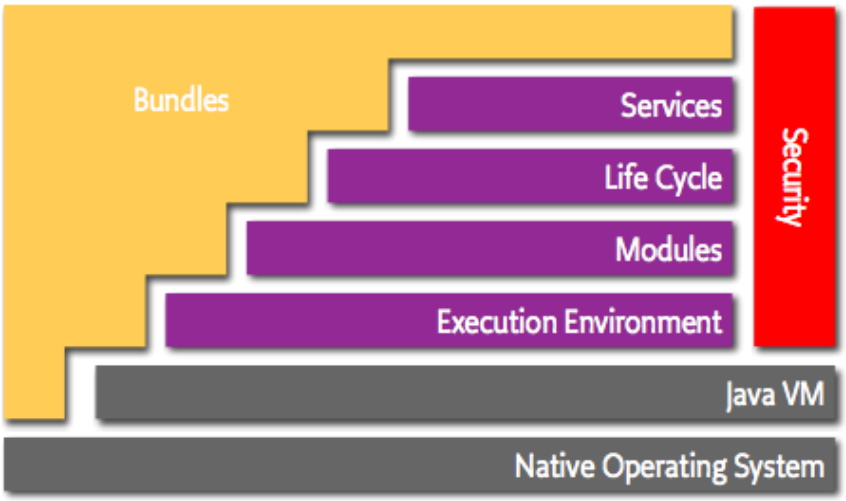

Figura 33. Capas de la especificación OSGi [1]

Los bundles realmente son archivos JAR y dentro de estos se manejan los **módulos**, **servicios** y **el ciclo de vida**.

En los **módulos** se ubica toda la implementación y código que contiene la funcionalidad del bundle. Esta codificación debería estar basada en un modelo modular en el que el comportamiento de cada uno es independiente al resto.

Este modelo es necesario ya que también en esta capa se definirá que código se podrá **importar o exportar** con respecto a otros bundles. Aquello que no se marque explícitamente como exportado o importado permanecerá escondido e inaccesible externamente. La exportación y la importación de código es una manera segura de comunicarse, pero no la única. OSGi da soporte a la codificación de aplicaciones orientada a servicios.

Los **servicios** brindan de dependencias dinámicas a través de la interacción de un registro de servicios. En este registro se pueden introducir servicios y así hacerlos accesibles

para que otros bundles puedan obtener dichos servicios y usarlos. En la figura 2 se muestra este comportamiento.

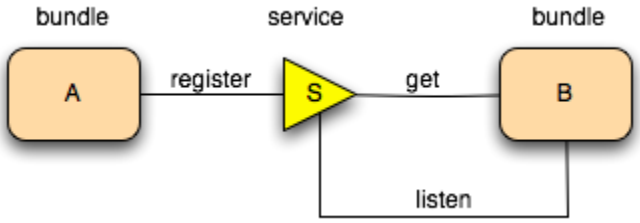

Figura 34. Esquema de orientación a servicios [1]

Por otra parte, los bundles funcionan bajo un **ciclo de vida**. En este flujo se puede intervenir y añadir código en las diferentes fases mediante una clase que se llamará Activator. Esto es conveniente para poder abrir conexiones a recursos al arrancar un bundle o para cerrarlas cuando se detenga, por ejemplo, en una base de datos.

El archivo **MANIFEST** también estará dentro del archivo JAR y contendrá información relativa al bundle como el nombre, identificador, paquetes exportados o importados etc. Todos los campos están definidos en la especificación OSGi.

El equipo de Eclipse pudo implementar todas estas especificaciones y llamó a su producto **Equinox**, el cual brindaba de toda esta infraestructura y algunas funciones adicionales no documentadas en OSGi. Una característica importante de este software es el uso de las extensiones y puntos de extensión, ya que nace como una alternativa a la programación orientada a los servicios.

Al desarrollar plugins (bundles en OSGi) para Eclipse, uno se da cuenta de que en todos los tutoriales y documentaciones hacen uso de los puntos de extensión como medio comunicativo entre los diferentes plugins. La diferencia entre estos y los servicios es la manera en que se comunican con el registro tanto la parte que registra el servicio como el que accede a él.

En los servicios participan tres entidades: consumidor, registro y proveedor.

- El **consumidor** se encarga de buscar la referencia del servicio a utilizar y con esa referencia obtenerlo del registro. Cuando lo desee lo usará llamando a las funciones de la API proporcionado por el servicio. Cuando ya no vaya a utilizarlo más se lo comunicará al registro mediante la referencia.

- En el **registro** se almacena las referencias y servicios.

- El **proveedor** implementa la funcionalidad del servicio, proporciona una API mediante una interfaz y creará una instancia la cual será registrada.

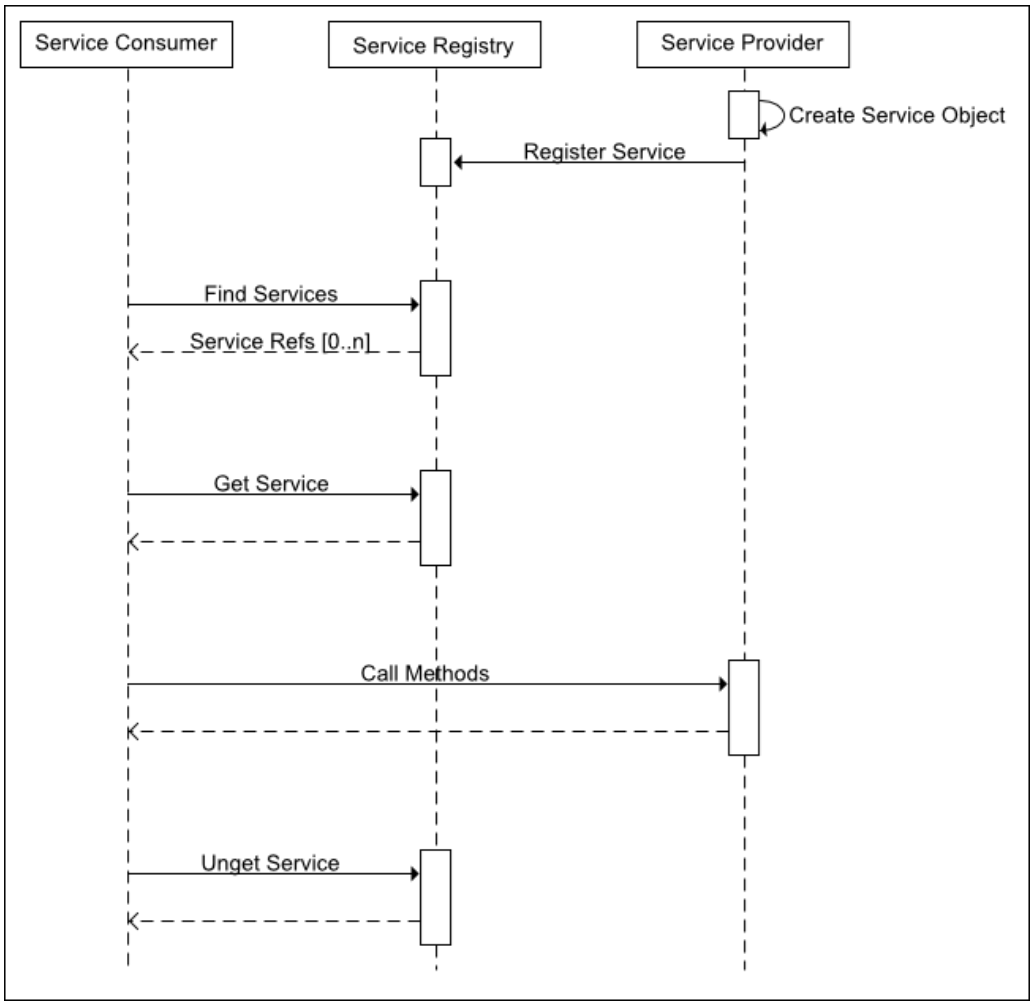

Figura 35. Diagrama de secuencia del registro y uso de un servicio. [2]

En las extensiones también participan tres entidades:

- El definidor del punto de extensión es el encargado de crear y habilitar un API para que otras extensiones puedan usar su funcionalidad. Una vez creado, procede a registrarlo o a declararlo en el registro de extensiones. Al ejecutar el definidor, este buscará todas las contribuciones (extensiones que hacen uso de su punto de extensión) ubicadas en el registro. En el momento en el que se vaya a utilizar la funcionalidad extendida, será el propio definidor quien instancie la contribución en el registro. Finalmente el registro le devolverá el objeto de la contribución para que pueda hacer uso de sus métodos y funcionalidad implementada.

- Tanto el definidor de punto de extensión como el contribuidor declararán sus implementaciones en un **registro.** Este nuevo agente no sólo se encargará de almacenar si no también será el que se comunique con el contribuidor a petición del definidor. En concreto, pedirá al contribuidor que le entregue una instancia del objeto que hace uso de la API del punto de extensión. Esta instancia será devuelta al definidor, el cual hará uso de ella.

- El **contribuidor** podrá declarar en el registro extensiones que hacen uso de puntos de extensión descritas por definidores. Estas extensiones implementadas por el contribuidor serán instanciadas por el registro y usadas por el definidor bajo demanda.

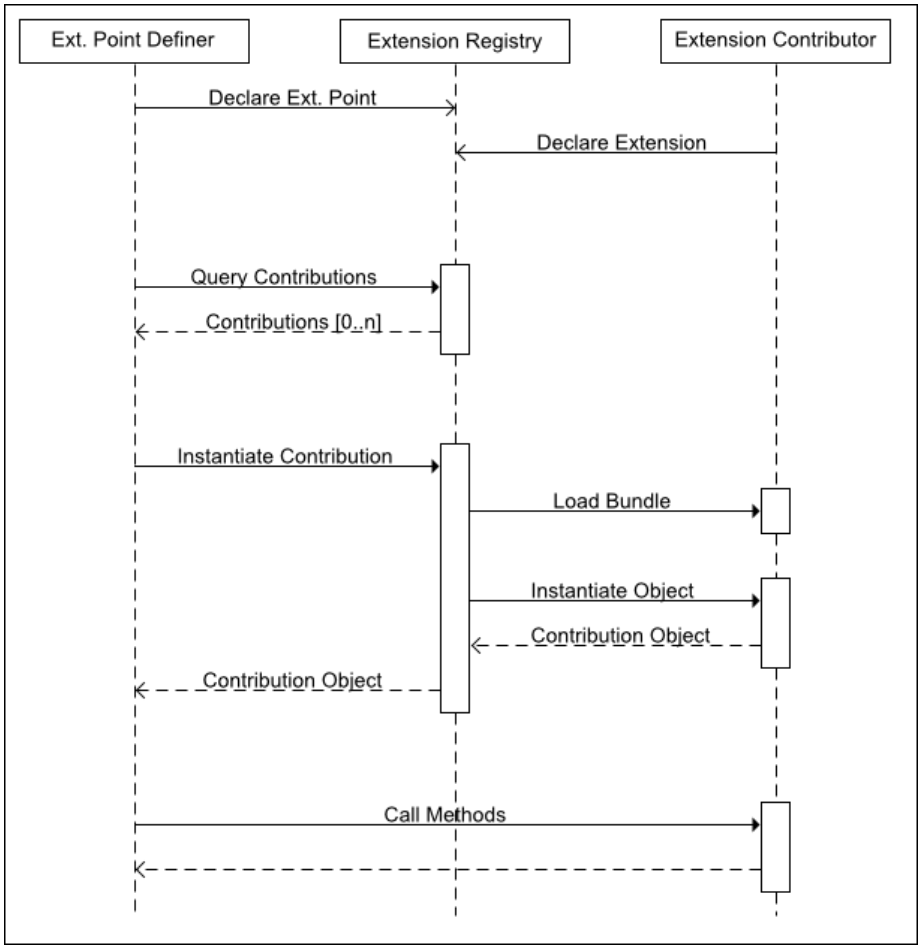

Figura 36. Diagrama de secuencia del registro y uso de una extensión. [2]

Para ilustrar este comportamiento imagínese que vamos a crear un plugin para Eclipse, el cual usa Equinox. En concreto vamos a añadir un botón en la barra de herramientas. Los desarrolladores de Eclipse ya se preocuparon de crear un plugin que manejara esta barra de herramientas y, para que desarrolladores de terceros extendieran su funcionalidad, declararon un punto de extensión que permite la incorporación de botones. En este caso nos especifican que tenemos que crear una clase que extienda de una interfaz y que implemente los métodos definidos por ella. Una vez realizado, declaramos nuestra extensión en el registro.
Cuando el usuario presione ese botón creado por nosotros lo que hará el registro será instanciar nuestra clase y dársela a ese plugin que maneja la barra de herramientas, la cual, a través de la interfaz, irá llamando a nuestros métodos.

Este sistema de extensiones se comporta de manera muy parecida a los servicios, pero a diferencia de estos, la extensión sólo instancia cuando se utiliza. Antes de instanciarse sólo se han registrado metadatos con formato "*xml*". Sin embargo, en el momento de

registrar un servicio se registra el servicio entero. Este y otros factores son los que diferencian los servicios de las extensiones.

Para finalizar, en la figura 5 se muestra los principales módulos en los que está compuesto Eclipse. Se pueden observar elementos que describiremos en siguientes apartados y también, bajo el nombre de *"New Tool"*, los plugins que podremos crear.

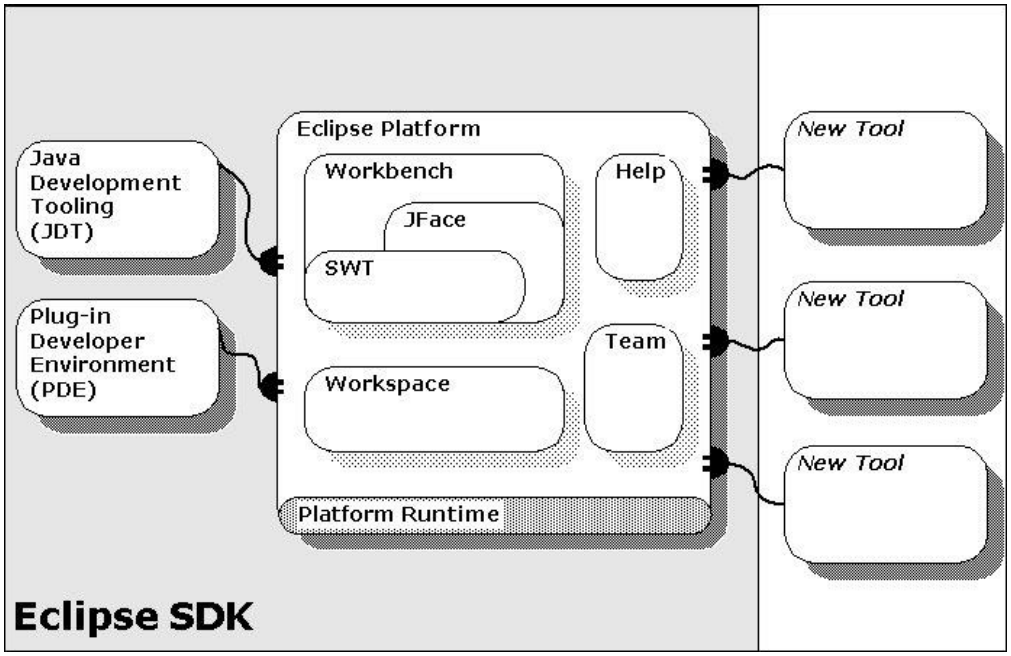

Figura 37. Arquitectura de la plataforma Eclipse [1]

D Uso de SWT y JFace en Eclipse

Para poder crear una interfaz gráfica se debe instanciar la clase Display, la cual inicializará los recursos y comunicaciones necesarios con el sistema operativo para manejar sus componentes gráficos. Sobre el Display se instanciará la clase Shell, encargada del manejo de maximizar, minimizar, cerrar, redimensionar, etc. La ventana principal sobre la que crearemos y combinaremos nuestros componentes gráficos o widgets.

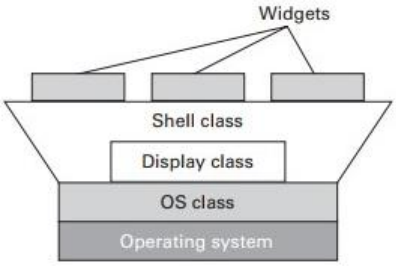

Figura 38. Estructura de comunicación de una interfaz gráfica SWT [3]

En Eclipse las clases Display y Shell ya vienen dadas de por sí, ya que la parte gráfica de este entorno de programación está desarrollada mediante SWT. Cuando vayamos a crear nuestros plugins, nuestro contexto será un Composite sobre el que organizaremos nuestros widgets.

Widget es la clase de la cual heredan todas las demás. Es la encargada de que se muestre la información correcta y la interfaz gráfica para el usuario. Posee unos métodos generales para el control del widget. Ya que cada sistema operativo puede proveer diferentes componentes, SWT maneja solo un subconjunto de ellos. Las principales subclases de Widget se muestran en la **figura 7**.

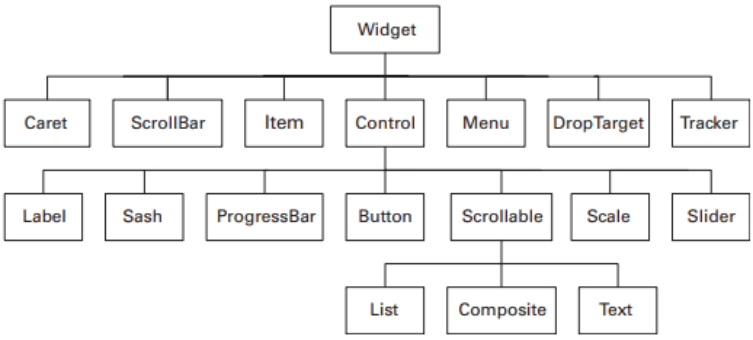

Figura 39. La clase Widget y sus principales subclases [3]

Los widgets más comunes tales como Label, Text, Button etc. heredan de la clase Control puesto que proporciona métodos que usan manejadores o *handles* para configurar estos componentes.

En particular, el widget composite permite insertar o agrupar widgets en él. Con este componente se pueden crear *layouts.* Los diferentes *composites* se muestran en la figura 8.

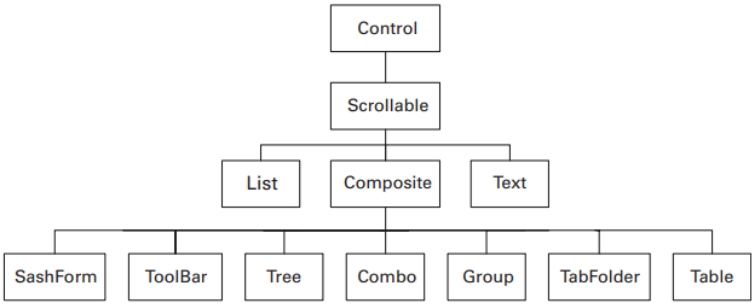

Figura 40. La clase Control y sus principales subclases [3]

Algunas de estas agrupaciones las usaremos para diseñar algunas de las vistas del plugin.

Por otra parte, a las unidades de JFace se las suele llamar adaptadores las cuales se enfocan en ciertas funcionalidades según su tipo:

- **Viewers:** Separan el aspecto visual del widget con la información contenida.
- **Actions y contributions:** Manejan y organizan los procesos de los eventos.
- **Registros de imágenes y fuentes:** Se encargan del alojamiento en memoria de imágenes y fuentes en un sitio común y organizado.

Dialogs y wizards: Aportan más funcionalidad a los diálogos de SWT para una mejor interacción con el usuario.